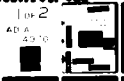
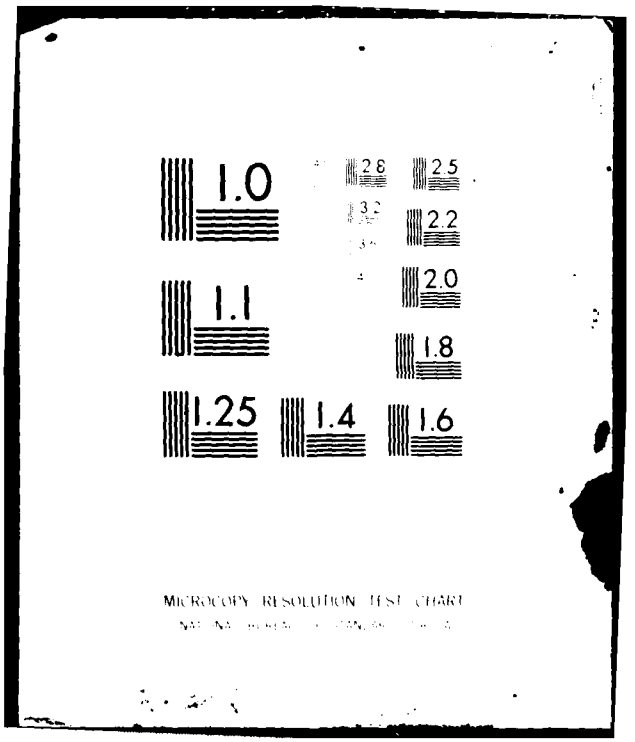
AD-A114 970 TENNESSEE UNIV KNOXVILLE DEPT OF ELECTRICAL ENGINEERING F/G 12/1
THE REPRESENTATION OF DISCRETE FUNCTIONS BY DECISION TREES.(U)
FEB 82 R C GONZALEZ, M G THOMASON, B M MORET N00014-78-C-0311
UNCLASSIFIED TR-EE/CS-82-20 NL




1.0
2.8
2.5
3.2
2.2
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

(12)

ELECTRICAL ENGINEERING DEPARTMENT

UNIVERSITY OF TENNESSEE

KNOXVILLE
TN 37916

(12)

THE REPRESENTATION OF DISCRETE FUNCTIONS BY DECISION TREES

Annual Report

Prepared for the

Office of Naval Research
Arlington, VA 22217

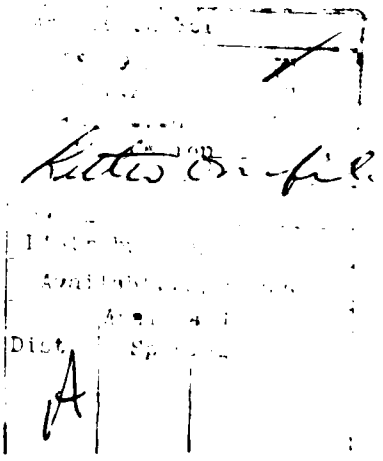
Contract No. N00014-78-C-0311

February 28, 1982

ABSTRACT

As applications of digital systems continue to expand, the need arises for better methods of analysis of functions of discrete variables. Particularly important is the ability to gauge accurately the difficulty of a problem; this leads to measuring a function's complexity. This in turn requires an implementation-independent model of function evaluation, one that also shows the contribution of individual variables to the function's complexity.

One such model, called a decision tree, is introduced; it is essentially a sequential evaluation procedure where, at each step, a variable's value is determined and the next action chosen accordingly. Decision trees have been used in switching circuits, data bases, pattern recognition, machine diagnosis, and remote data processing. The activity of a variable, a new concept that measures the contribution of a variable to the complexity of a function, is defined and its relation to decision trees is described. Based upon these results (which can be generalized to recursive functions and hierarchies of relations), a complexity measure is proposed. The use of that measure and of the concept of activity in testing large systems (where a number of variables may be inaccessible) is then examined, with particular [illegible] on continuous checking of systems in operation.



The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Office of Naval Research, the Department of Defense, or the U.S. Government.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: TR-EE/CS-82-20	2. GOVT ACCESSION NO.: AD-A114 976
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): The Representation of Discrete Functions by Decision Trees	5. TYPE OF REPORT & PERIOD COVERED: Annual Report Feb. 1, 1981-Jan. 31, 1982
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): R. C. Gonzalez, M. G. Thomason, B. M. E. Moret	8. CONTRACT OR GRANT NUMBER(s): N00014-78-C-0311
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Electrical Engineering Department, University of Tennessee, Knoxville, TN 37990	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA 22217, Code 414, Contract Monitor: Ms. P. Papantoni-Kazakos	12. REPORT DATE: Feb. 28, 1982
	13. NUMBER OF PAGES: 125
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same as 11	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): Distribution Unlimited	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report): Same as 16	
18. SUPPLEMENTARY NOTES: None	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number): Activity, Decision Diagrams, Decision Tables, Decision Trees, Testing, Cost, Sequential, Digital.	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

As applications of digital systems continue to expand, the need arises for better methods of analysis of functions of discrete variables. Particularly important is the ability to gauge accurately the difficulty of a problem; this leads to measuring a function's complexity. This in turn requires an implementation-independent model of function evaluation, one that also shows the contribution of individual variables to the function's complexity.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

One such model, called a decision tree, is introduced; it is essentially a sequential evaluation procedure where, at each step, a variable's value is determined and the next action chosen accordingly. Decision trees have been used in switching circuits, data bases, pattern recognition, machine diagnosis, and remote data processing. The activity of a variable, a new concept that measures the contribution of a variable to the complexity of a function, is defined and its relation to decision trees is described. Based upon these results (which can be generalized to recursive functions and hierarchies of relations), a complexity measure is proposed. The use of that measure and of the concept of activity in testing large systems (where a number of variables may be inaccessible) is then examined, with particular emphasis on continuous checking of systems in operation.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

As applications of digital systems continue to expand, the need arises for better methods of analysis of functions of discrete (in particular, binary) variables. Such functions often represent the total knowledge available to the designer or engineer about a system, so that their understanding is critical to the development of design methods, cost estimates, troubleshooting procedures, etc.

Of particular importance in this regard is the ability to gauge accurately the difficulty of the problem. This enables the system designer to evaluate the feasibility of the project, select tools or methods of appropriate capabilities, and evolve cost, time, and other estimates. Thus, a crucial part of the analysis is measuring complexity, more precisely, a function's complexity.

Such a measure can be defined experimentally, thereby relating it directly to human experience, as is the case in software science (Halstead 77). The development can also be analytical, based upon some model of function evaluation; such measures often quantify the expense of some resource present in the model, such as time and memory in concrete complexity theory (Aho 74, Garey 79), or logic gates in combinational complexity theory (Savage 76, Pippenger 77). These three approaches, however, assume further knowledge about the system because they are not implementation-independent. Moreover, they do not

explicitly show the contribution of individual input variables to the complexity of the function.

An inplementation-independent model of function evaluation is also needed in computational complexity theory in order to establish lower bounds on the amount of work required for evaluation. One such model, which also allows an analysis of the contribution of individual variables, is called a decision tree; it has been used to prove lower bounds on sorting (Knuth 71), set manipulation (Reingold 72), and recognition of graph properties (Rivest 76b). A decision tree is essentially a sequential evaluation procedure whereby the value of a variable is determined and the next action (choice of another variable to evaluate or decision as to the value of the function) chosen accordingly. Figure 1.1 shows a decision tree for sorting three elements. The variables are all $\binom{3}{2} = 3$ possible comparisons between two elements and are binary in that the result of comparing (a:b) is either ($a \geq b$) or ($a < b$). Thus, each internal node of the tree has two children, corresponding to the two possible values of the variable evaluated at that node. The external nodes are values of the function, in this case permutations.

Since decision trees are models of sequential evaluation, they have been used extensively wherever parallel or tabular data must be converted to sequential procedures, as in decision tables (Metzner 77), switching theory (Lee 59, Cerny 79), machine diagnosis (Chang 70), and data base queries (Ullman 80), or when inputs are provided one at a time, as in taxonomy (Jardine 71, Garey 72), multistage pattern recognition

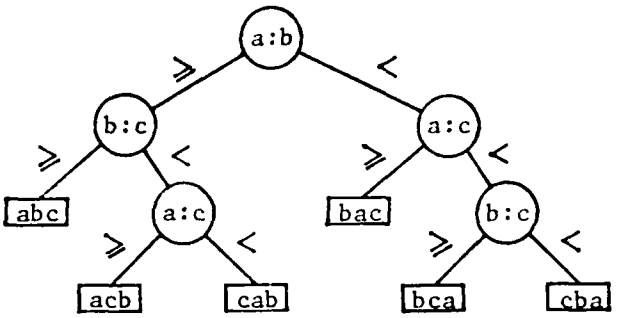


Figure 1.1. A decision tree for sorting three elements.

(Sethi 77), and remote data processing (You 76). As a consequence, decision trees have come to be known under many names and disconnected results about them appear under many guises in the technical literature.

This work generalizes and unifies the concept of a decision tree, presenting the first formal definition of it. Known results are reviewed and the various measures used to characterize decision trees are discussed. In particular, a comprehensive analysis of the computational complexity of these measures is presented, including some new results on the worst case testing complexity of Boolean functions. The activity of a variable, a new concept that measures the contribution of a variable to the complexity of a function, is defined and its relation to decision trees is described. These results are subsequently generalized to relations and recursive functions. Based upon these developments, a complexity measure for functions of discrete variables is proposed and its use in testing large digital systems is examined. The exposition concludes with an assessment of the work done and recommendations for future research.

CHAPTER 2

PRELIMINARIES

2.1. Introduction

As mentioned in the previous chapter, decision trees have been used in a number of areas, including computer science, biology, engineering, and management, with diverse terminology and degree of generality. The purpose of this chapter is to provide some basic definitions and results and to establish a unified terminology in which to express the general problem as well as the various special cases encountered in the literature.

Some elementary concepts from Boolean algebra, graph theory, and concrete complexity theory are first reviewed, as they will be used throughout the following chapters. Decision trees and diagrams are then defined, starting with the simplest and most widely encountered family of functions--the Boolean functions--and generalizing to (partial) functions of discrete variables. The special cases of decision tables and identification (taxonomy), which have attracted more attention from researchers than any other aspect of the overall problem, are then examined within the established framework.

2.2. Basic Concepts

2.2.1. Boolean functions

The following definitions and results can be found in any textbook on Boolean algebra or switching theory (Harrison 65, Friedman 75).

A (completely specified) Boolean function of n variables, $f(x_1, \ldots, x_n)$, is a mapping from $\{0, 1\}^n$ to $\{0, 1\}$, where $\{0, 1\}^n$ denotes the n-fold cartesian product of $\{0, 1\}$, that is, the set of binary n-tuples. The set of all n-tuples mapped by the function to the value 1, $\{(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n) = 1\}$, is called the set of minterms of f. A Boolean function can be specified by describing the mapping (giving its "truth table") or by listing its minterms; it can also be represented by a Boolean formula, usually in terms of the three operations of disjunction (+), conjunction (·), and complementation ($\bar{}$). An important canonical representation as a formula is the disjunctive normal form (DNF), formed by a disjunction of conjunctions, where each conjunction includes all variables and represents a minterm; this form can often be simplified by combining conjunctions to obtain a sum-of-products (SOP), which can be minimized (with respect to the number of conjunctions) by the well-known Quine-McCluskey algorithm.

A function of n variables can be expressed in terms of two functions of n - 1 variables by means of Shannon's expansion theorem:

$$f(x_1, \ldots, x_n) = \overline{x_i} \cdot f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) + x_i \cdot f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) ,$$

for each choice of x_i, $1 \leq i \leq n$; this will be written

$$f(x_1, \ldots, x_n) = \overline{x_i} \cdot f_{(x_i=0)} + x_i \cdot f_{(x_i=1)} .$$

A function, $f(x_1, \ldots, x_n)$, is <u>fictitiously dependent</u> upon variable x_i, $1 \leq i \leq n$, (or x_i is a <u>redundant</u> variable for f) exactly when $f_{(x_i=0)} = f_{(x_i=1)}$. In particular, a function is fictitiously dependent on each of its variables exactly when it is a constant; a function with no redundant variables is called <u>intrinsic</u>. Since the domain of a Boolean function of n variables has cardinality 2^n and its range has cardinality 2, there exist 2^{2^n} distinct Boolean functions of n variables; it is easily shown that almost all (in the sense of asymptotics[†]) Boolean functions are intrinsic.

<u>Example 2.1.</u> Let the Boolean function of three variables, $f(x_1, x_2, x_3)$, be given by the mapping:

f:	(0, 0, 0) → 0	(1, 0, 0) → 0
	(0, 0, 1) → 0	(1, 0, 1) → 0
	(0, 1, 0) → 1	(1, 1, 0) → 0
	(0, 1, 1) → 1	(1, 1, 1) → 1

This function can also be represented by the list of its minterms:

$$\{(0, 1, 0), (0, 1, 1), (1, 1, 1)\} ,$$

or by its disjunctive normal form:

$$f(x_1, x_2, x_3) = \overline{x_1}x_2\overline{x_3} + \overline{x_1}x_2x_3 + x_1x_2x_3 ,$$

[†]That is, the ratio of the number of items of interest to the total number of items tends to 1 as the total number of items grows.

which can be simplified to yield the minimum sum-of-products form:

$$f(x_1, x_2, x_3) = \overline{x_1}x_2 + x_2x_3 \ .$$

It is easily verified that this function is intrinsic. □

2.2.2. Trees

The following definitions and results are standard topics in graph theory (Harary 69) and computer science (Knuth 73).

A (finite) graph, G = (V, E), consists of a (finite) set of vertices (or nodes), V, together with a set, E, of unordered pairs of distinct vertices from V, called edges; if E is a multiset (i.e., elements may be repeated), then G is called a hypergraph. A graph is called directed (is a digraph) if each edge is an ordered pair; E is then considered as an irreflexive relation on V × V. Let S be a set of symbols; a graph is vertex labelled if there is a function, g: V → S; it is edge labelled if there is a function, h: E → S.

Let e = (v_1, v_2) be an edge; then e and v_1 (and e and v_2) are said to be incident; v_1 and v_2 are called adjacent vertices. The degree of a vertex in a graph is the number of edges incident with it; in a digraph, the degree of a vertex is the sum of the in-degree (the number of edges directed towards the vertex) and the out-degree (the number of edges directed away from the vertex). A cycle is an alternating sequence of three or more vertices and edges, v_0, e_1, v_1, e_2, ..., v_{n-1}, e_n, v_0, beginning and ending at the same vertex, in which each edge is incident with the two vertices immediately preceding or following it, and such

that no two vertices are identical. A graph without cycles is called acyclic.

A tree is a connected acyclic graph; it is easily shown that a tree with n vertices must have exactly n - 1 edges. A rooted tree is a tree with a distinguished vertex called the root. It is often convenient to have a definition of a tree that introduces more structure and lends itself to inductive proofs; a tree is therefore defined recursively as a finite nonempty set of vertices such that there is a distinguished vertex called the root and the remaining vertices are partitioned into zero or more disjoint sets, each in turn a tree (called a subtree). Such a tree can be thought of as a connected directed acyclic graph where all edges are directed away from the root.

Thus the in-degree of the root is zero and that of every other vertex is one. Vertices with nonzero out-degree are called internal; those with zero out-degree are called external vertices or leaves. A full k-ary tree is one where every internal vertex has an out-degree of k. [This terminology differs somewhat from that of (Knuth 73)]. Vertices adjacent to the root are called its children and the root is their parent. Since a single path exists from the root to any vertex in the tree, the depth of a vertex is defined as the number of edges traversed on the path; the height of a tree is defined as the maximal depth of any vertex in the tree; finally, the path length of a tree is the sum of the depths of its leaves.

When the order of the children of each vertex is of importance, the tree will be called ordered or planar (since the manner of imbedding the

tree in a plane is then relevant). Following the convention in use in computer science, trees will be drawn with the root at the top; in ordered trees, subtrees will be drawn left to right.

Example 2.2. The tree of Figure 1.1 (page 3) is an ordered, vertex and edge labelled, full binary tree. The root is labelled (a:b) and is at depth 0; the leaves are labelled by the permutations of (abc) and are at depths 2 and 3; the height of the tree is 3. □

2.2.3. Concrete complexity

This section is based on (Garey 79) and uses the same terminology. Concrete complexity theory is concerned with measuring the computational complexity of algorithms, usually in terms of time and space.

An algorithm is a precise, step-by-step procedure (e.g., a computer program) for solving a problem. A problem is composed of parameters of unspecified value and a question to be answered, and is specified by describing the nature of its parameters and the properties that its solution must possess. An instance of a problem is obtained by providing specific values for the problem parameters. An algorithm solves a problem if it is guaranteed to provide a solution when applied to any instance of the problem.

Example 2.3. The following problem is well known as the minimum cover problem. The problem parameters are a set, S, and a collection, C, of subsets of S (i.e., $C \subseteq 2^S$); the problem question asks to find the smallest subset, C′, of C such that C′ is a cover for S, that is, each

element of S belongs to at least one element of C′. One instance of the problem is given by S = {a, b, c, d}, C = {{a}, {a, c}, {a, b, d}, {b}, {d}}, for which the solution is C′ = {{a, c}, {a, b, d}}. An algorithm to solve this problem must, for each problem instance, find the minimum cover, C′, or report that no such cover exists (which happens whenever C itself is not a cover). □

Concrete complexity theory concentrates on measuring the time requirements of an algorithm on some reasonable model of computation, such as a Turing machine, a register machine, or a general purpose computer. These requirements are expressed as a function of the size of the problem instance, that is, in a sense, as a function of the size of the input to the algorithm. The size of a problem instance is measured by encoding the instance in a reasonable manner (that is, in a manner that is not artificially wasteful of space) and measuring the length of the code.

Example 2.4. An instance of the minimum cover problem can be encoded in binary by first giving the size of S, which takes about $\log_2 |S|$ symbols, then coding each element of C by $|S|$ digits, where the i-th digit is 1 if the i-th element of S belongs to that element of C and is 0 otherwise. This in turn takes $|C| \cdot |S|$ symbols, so that the input has size $\log_2 |S| + |C| \cdot |S|$, which is of the order of $|C| \cdot |S|$ for large sets. □

The time complexity function expresses the time requirements of an algorithm by giving, for the size of each instance, the maximum amount of time spent by the algorithm to solve a problem instance of that size.

Since different encodings will result in somewhat different size measures, the function is usually expressed as the order of the rate of growth of the time requirements. Specifically, a function, f(n), is said to be $\mathcal{O}(g(n))$ whenever there is a constant, c, such that $|f(n)| \leq c \cdot |g(n)|$ for all values of n. Thus, for instance, $3n + 5$ is $\mathcal{O}(n)$, $4n^2 + n$ is $\mathcal{O}(n^2)$, and $4e^n + n^{10}$ is $\mathcal{O}(e^n)$.

A polynomial time algorithm has time complexity no larger than $\mathcal{O}(p(n))$, where n is the input size and p is some polynomial function; when the time complexity of an algorithm cannot be so bounded, the algorithm is said to require exponential time. Polynomial time is associated with efficient, and exponential time with inefficient, algorithms, as is illustrated in Table 2.1, where running times are tabulated for several time complexity functions and instance sizes, assuming each step to take one microsecond (1 μs) on present day computers (the upper numbers) and one picosecond (1 ps) on futuristic machines (the lower numbers). Not only are exponential time algorithms incomparably slower than polynomial time ones, but futuristic machines bring only minor relief, whereas they considerably speed up polynomial time algorithms.

It is sometimes possible to show that a problem cannot be solved in less than $\mathcal{O}(f(n))$ time, for some function f. A well-known example is sorting, which is known to require at least $\mathcal{O}(n \cdot \log n)$ comparisons for n objects. Often, however, such results cannot be attained. In particular, there exist numerous problems for which only exponential time algorithms are known, but which are not known to require this time

Table 2.1
Polynomial Time Versus Exponential Time

Size: \ Complexity:	$O(n)$	$O(n^2)$	$O(n^3)$	$O(n^5)$	$O(2^n)$	$O(3^n)$
5	5 μs	25 μs	125 μs	3.1 ms	32 μs	243 μs
	5 ps	25 ps	125 ps	3.1 ns	32 ps	243 ps
10	10 μs	100 μs	1 ms	100 ms	1 ms	59 ms
	10 ps	100 ps	1 ns	100 ns	1 ns	59 ns
100	100 μs	10 ms	1 s	2.8 hrs	$4 \cdot 10^{14}$ cent.	$2 \cdot 10^{32}$ cent.
	100 ps	10 ns	1 μs	10 ms	$4 \cdot 10^{8}$ cent.	$2 \cdot 10^{26}$ cent.
1000	1 ms	1 s	16.6 min	31.7 yrs	10^{285} cent.	10^{461} cent.
	1 ns	1 μs	1 ms	16.6 min	10^{279} cent.	10^{455} cent.

complexity. (Indeed, very few problems of practical importance have been shown to require exponential time, unless the solution itself takes exponential time to describe.)

Of particular interest among the latter are problems belonging to the class NP (for nondeterministic polynomial); those are all the problems, the solutions of which can be verified in polynomial time. A nondeterministic machine can solve an NP problem by "guessing" a structure and verifying that it is a solution in polynomial time. In particular, of course, all problems solvable in polynomial time (the class P) are in NP. One of the most important open questions in computer science is to decide whether P = NP or not. (The available evidence is discouraging; many of the NP problems for which no polynomial time algorithm is known are of great practical importance and have received a lot of attention over the past thirty years, but to no avail.)

Example 2.5. The so-called decision problem for the minimum cover has the same parameters as the minimum cover problem, plus a constant, $k < |C|$. The question is: does there exist a subset C' of C such that C' is a cover for S and $|C'| \leq k$? This problem is in NP since a nondeterministic machine can "guess" a subset C' and verify in polynomial time whether $|C'| \leq k$. Clearly, this problem is a special case of the minimum cover problem, since the minimum cover itself must be a solution to the decision problem, if any solution exists. □

As part of the effort to solve the question of whether P = NP, researchers identified a class of problems that are complete for the

set NP. That is, if any of these problems can be solved in polynomial time, then so can all problems in NP; thus, in a sense, these problems, called NP-complete, are the hardest problems in NP. The decision problem for the minimum cover is an example of an NP-complete problem (Karp 72).

Since the analysis leading to the definition of NP-complete problems was done in terms of language membership, all NP-complete problems are decision problems, that is, they ask a question about the existence of a particular structure. The concept, however, can be enlarged to optimization problems, as illustrated below for the minimum cover problem.

As seen above, the decision problem for the minimum cover is NP-complete and is a special case of the minimum cover problem; thus, in a sense, the latter is at least as hard as the hardest problems in NP. Such a problem is called NP-hard. However, as will be seen, the minimum cover problem is no harder than NP-complete problems in the sense that, if its associated decision problem has a polynomial time solution (i.e., if P = NP), then that solution can be used to solve the minimum cover problem in polynomial time. Such problems are then called NP-easy; a problem that is both NP-hard and NP-easy is termed NP-equivalent.

That the minimum cover problem is NP-easy can be seen by using the standard technique of an intermediate completion problem. The completion problem for the minimum cover has the same parameters as the decision problem, plus a "partial solution" subset, $C'' \subseteq C$; the question is: does there exist a subset, C', of C such that $|C'| \leq k$, C' is a cover

for S, and $C' \supseteq C''$ (C' "completes" C'')? This problem is clearly in NP; moreover, the decision problem is a special case of it, where C'' is chosen as the empty set. Hence the completion problem in NP-complete. Suppose now that P = NP, so that both the decision and the completion problems have polynomial time solutions. Since the cardinality of the solution set, C', is an integer between 0 and $|C|$, the decision problem can be used $\log_2 |C|$ times, in a binary search over the interval, to determine if a minimal solution exists and, if so, its cardinality, k_{min}; this clearly takes polynomial time. The completion problem can then be used, with k set to k_{min}, to build the solution set element by element as follows. First, let C'' include a single element of C; for at least one choice of element, C'' can be completed; keeping that element, let now C'' include one other element of C; the process continues until all k_{min} elements have been found, using the completion problem at most

$$|C| + (|C|-1) + \ldots + (|C|-k_{min}+1)$$

$$= k_{min} \cdot (2 \cdot |C| + 1 - k_{min})/2$$

times, again a clearly polynomial time process. Thus, the minimization problem can be solved in polynomial time if the decision problem can; hence it is NP-easy.

2.3. Decision Trees and Diagrams

2.3.1. The case of completely specified Boolean functions

Definition 2.1. Let $f(x_1, \ldots, x_n)$ be a (completely specified) Boolean function. If f is a constant, then the decision tree for f consists of a single vertex labelled by that constant. Otherwise, for each x_i, $1 \leq i \leq n$, f has a decision tree composed of a root labelled x_i and two decision subtrees, the first for the subfunction $f_{(x_i=0)}$, the second for the subfunction $f_{(x_i=1)}$. □

Thus, decision trees for Boolean functions are explicit illustrations of Shannon's expansion theorem. This recursive definition closely parallels that given for trees in Section 2.2.2; it defines decision trees for Boolean functions as rooted, ordered, vertex-labelled, full binary trees. (The choice of ordering rather than edge labelling to distinguish subtrees is arbitrary and a matter of convenience.) To an extent, this definition prevents redundant testing in a tree, in that no more testing may take place as soon as the function has been reduced to a constant.

The evaluation of a Boolean function represented by a decision tree starts by ascertaining the value of the variable associated with the root of the tree; it then proceeds by repeating the process, on the left subtree if the variable was false or on the right subtree if the variable was true, until a leaf is reached; the label of the leaf gives the value of the function.

Example 2.6. The Boolean function of Example 2.1 was given by the formula

$$f(x_1, x_2, x_3) = \overline{x_1}x_2 + x_2x_3 .$$

Two possible decision trees for that function are shown in Figure 2.1. Since decision trees are ordered, the left subtree of a node always corresponds to the variable associated with the node being evaluated at 0, the right subtree to the variable being evaluated at 1. Thus, evaluation on the tree of Figure 2.1(a) for the triple of values (0, 1, 0) would first examine variable x_1; on finding it to be 0, it would proceed to the left subtree, there to examine variable x_2; since $x_2 = 1$, the right subtree would next be used, thereby encountering a leaf and terminating the evaluation. The label of that leaf, 1, is the value of the function for the given triple of values, obtained by examining only two of the three variables; the same evaluation on the other tree would require that all three variables be examined. The first tree represents the expansion:

$$f(x_1, x_2, x_3) = \overline{x_1}\cdot(\overline{x_2}\cdot 0 + x_2\cdot 1) + x_1\cdot(\overline{x_3}\cdot 0 + x_3\cdot(\overline{x_2}\cdot 0 + x_2\cdot 1)) . \quad \square$$

Since the root of each subtree can be labelled with any of the untested variables, the number of possible decision trees for a given function is in general very large. For instance, the function of Example 2.6 has ten distinct decision trees, as depicted in Figure 2.2.

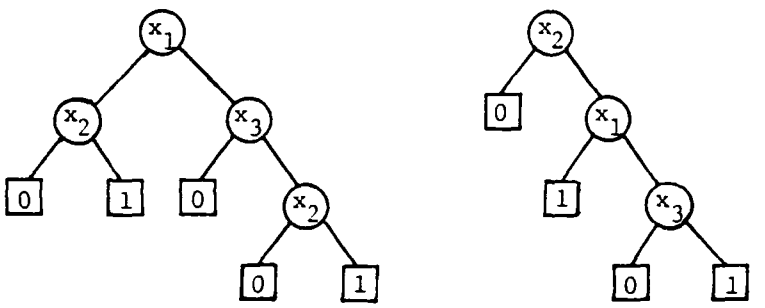


Figure 2.1. Two decision trees for the function of Example 2.6.

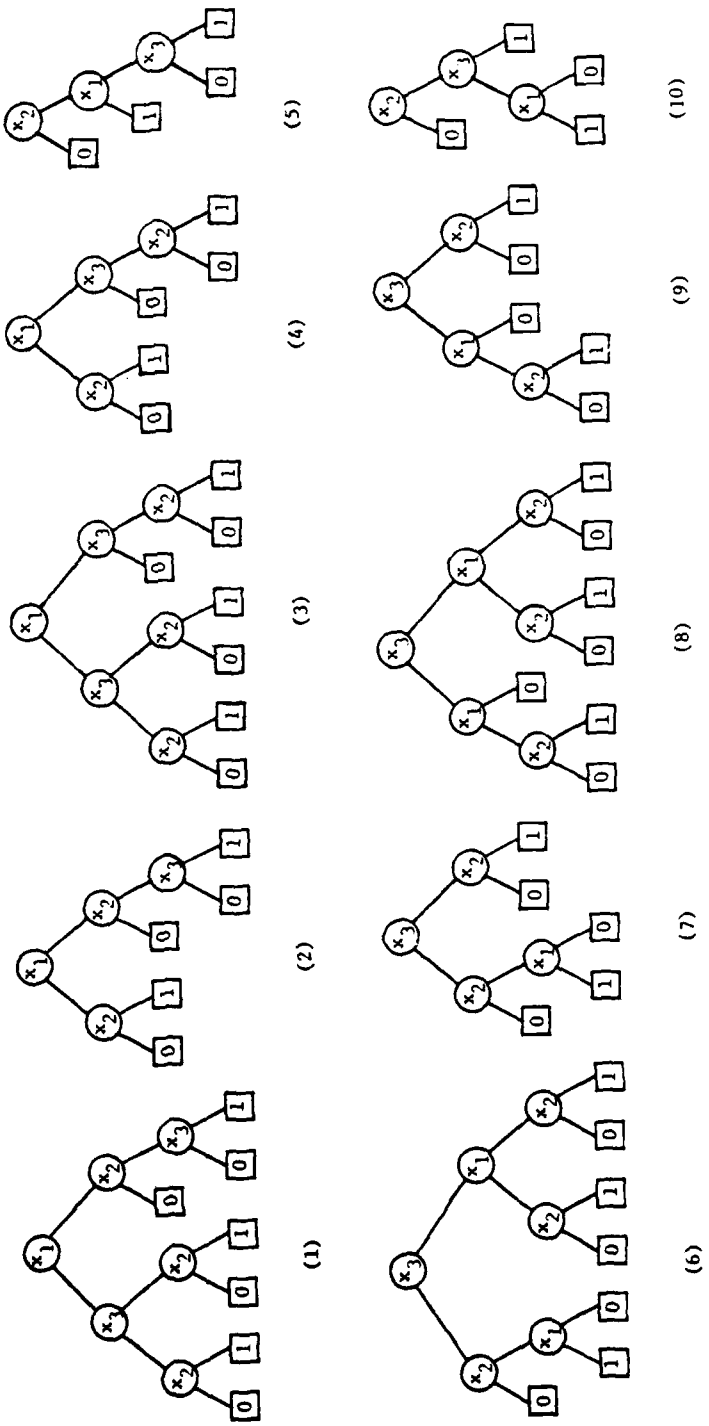


Figure 2.2. All ten possible decision trees for the function of Example 2.6.

In fact, it is easily seen that a Boolean function of n variables may have up to

$$\mathcal{N}_T(n) = \prod_{i=0}^{n-1} (n - i)^{2^i} \qquad (2.3.1)$$

distinct decision trees (n choices are possible for the root, followed by n - 1 choices on each of the two subtrees, or $(n - 1)^2$ choices; in general, up to $(n - k)^{2^k}$ choices are possible at depth k). This corresponds to the recurrence relation:

$$\mathcal{N}_T(n) = n \cdot (\mathcal{N}_T(n - 1))^2 , \qquad (2.3.2)$$

which shows that $\mathcal{N}_T(n)$ grows faster than 2^{2^n}. The first few values of $\mathcal{N}_T(n)$ are listed in Table 2.2.

Table 2.2
The Number of Decision Trees for Boolean Functions

n	$\mathcal{N}_T(n)$	n	$\mathcal{N}_T(n)$
1	1	6	$1.65 \cdot 10^{27}$
2	2	7	$1.91 \cdot 10^{55}$
3	12	8	$2.91 \cdot 10^{111}$
4	576	9	$7.64 \cdot 10^{224}$
5	1,658,880	10	$5.84 \cdot 10^{448}$

Not all decision tree representatives of a function are equally desirable. Thus, several criteria have been used in order to select an

appropriate representation; such criteria attempt to measure important properties of decision trees, in particular their realization and usage costs.

In the most general case, each variable has an associated testing cost (corresponding to the time needed for its evaluation, or the actual expense incurred at each evaluation, or some other cost related to the determination of the variable's value) and an implementation (or storage) cost (corresponding to the amount of hardware or memory necessary to choose a path depending on the variable's value or to some other cost related to the apparatus needed for decision making). If such costs are unknown, they are taken to be unity. (A third cost may arise in practice, corresponding to the cost of the hardware equipment needed to obtain a value for a variable, the sensor cost. This measure has no direct relation to the tree; it is a onetime only cost, incurred as soon as a variable is tested somewhere in the tree, and is significant only for redundant variables; it will be further discussed in Chapters 3 and 4.)

Based on this information, several measures have been defined on decision trees.

Definition 2.2.

(a) The storage cost, α, of a decision tree is the sum of the implementation costs of its nodes.

(b) The worst case testing cost, h, of a decision tree is the maximum, taken over all the paths from the root to the leaves, of the path

costs (where the cost of a path is the sum of the testing costs of the variables examined on that path).

(c) The total testing cost, η, of a decision tree is the sum, taken over all the paths from the root to the leaves, of the path costs.

(d) The normalized testing cost, H, of a decision tree is the total testing cost divided by the number of leaves. □

When costs are unity, the storage cost reduces to the number of internal nodes in the tree, the worst case testing cost reduces to the height of the tree, while the total testing cost reduces to the path length of the tree and the normalized testing cost reduces to the average path length of the tree. The path length of the tree is itself a special case of the tree path entropy defined in (Green 73) and the average path length a special case of the normalized tree path entropy.

Example 2.7. Assume the following costs for the function of Example 2.6.

storage costs	x_1: 1	x_2: 2	x_3: 3
testing costs	x_1: 5	x_2: 2	x_3: 6

The various measures defined above are then computed for the two trees of Figure 2.1 (page 18) and listed below.

measure	tree (a)	tree (b)	measure	tree (a)	tree (b)
α	8	6	node count	4	3
h	13	13	height	3	3
η	51	35	path length	12	9
H	10.2	8.75	av. path length	2.4	2.25

□

A given decision tree is likely to have common subtrees; those can then be constructed just once and used on other paths of the tree, instead of being duplicated throughout. The structure thus created is not a tree, since the in-degree of some nodes may be greater than one; it may be assimilated to a vertex and edge labelled, directed, acyclic hypergraph where all the nodes have out-degree zero or two and where there is a single node of in-degree zero (the root). This defines a decision diagram; further requiring that there be only one leaf labelled 1 (the "finish" node) yields a free Boolean graph.

By definition, then, a decision diagram is associated with a decision tree; there is a one-to-one correspondence between the paths in the tree and those in the diagram. Thus, all the measures of Definition 2.2 are valid on decision diagrams, taking the same values as on their associated trees. However, the tree storage cost, α, is inadequate since it does not describe the savings resulting from the identification of common subtrees; it is replaced by the diagram storage cost, β, which is defined as the sum of the implementation costs of the diagram's nodes. When costs are unity, this reduces to the number of internal nodes of the diagram.

Example 2.8. Consider again the function of Example 2.6; two free Boolean graph representations, associated with the corresponding trees of Figure 2.1 (page 18), are depicted in Figure 2.3. Diagram (a) has $\beta = 6$ and so does diagram (b); both diagrams have three internal nodes, in contrast to the corresponding trees. It is clear from Figure 2.3

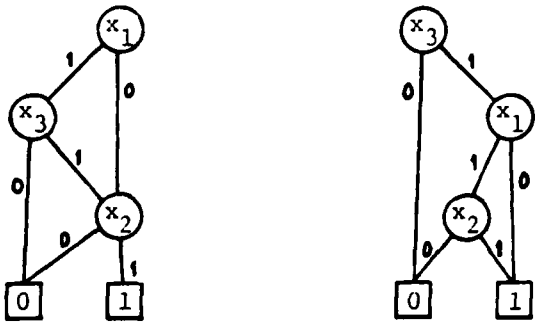


Figure 2.3. The two free Boolean graphs of Example 2.8.

that edge labelling is preferable to (subdiagram) ordering as a means to distinguish edges, since diagrams have a decidedly more complex structure than trees. □

Further information is often available about a function in the form of a probability distribution on the variables' values, that is, a function $p: \{0, 1\}^n \to [0, 1]$; when the distribution is not specified, it can be taken to be uniform, that is, each n-tuple of values is equally likely. This distribution allows a quantitative measurement of the average behavior of a decision tree or diagram.

Specifically, a path from the root to some node can be assigned a probability, which is simply the sum of the probabilities of all combinations of values that can lead to that node; thus, each path has an expected testing cost, which is the product of its probability times its cost. The expected testing cost, E, of a decision tree or diagram is then defined as the sum, taken over all the paths from the root to the leaves, of the expected path testing costs.

When all costs are unity and the distribution of variables' values is uniform, the information necessary and sufficient to compute all of the various tree measures defined above consists of the number of leaves at each depth [or, equivalently, the number of internal nodes at each depth, since one set can easily be computed from the other (Knuth 73)]. Thus a decision tree for a function of n variables can be entirely characterized by an (n + 1)-tuple, $(\lambda_0, \lambda_1, \ldots, \lambda_n)$, where λ_i is the number of leaves at depth i; this notation will be called leaf profile, by analogy with a similar notation introduced in (Miller 79). The five tree measures defined above can then be rewritten as simple functions of the leaf profile:

$$\text{node count, } \alpha = \sum_{i=0}^{n} \lambda_i - 1 \ ;$$

$$\text{height, } h = \max \{ i \mid \lambda_i \neq 0 \} \ ;$$

$$\text{path length, } \eta = \sum_{i=0}^{n} i \cdot \lambda_i \ ;$$

$$\text{average path length, } H = \eta / \sum_{i=0}^{n} \lambda_i \ ;$$

$$\text{expected number of tests, } E = \sum_{i=0}^{n} i \cdot 2^{-i} \cdot \lambda_i \ .$$

The leaf profile provides more than a convenient shorthand for simple problems; it allows a lexicographic ordering of decision trees. This, in turn, gives rise to two other measures, the maximum profile, which ranks as best that tree which is largest in lexicographic order (on the

grounds that leaves should be encountered as soon as possible), and the minimum reverse profile, which ranks as best that tree which is smallest in reverse lexicographic order (on the grounds that long paths should be minimized). Both measures are easily generalized to nonuniform probability distributions (by replacing "number of leaves" by "probability of leaves") but cannot be applied when nonunity costs are present.

Example 2.9. Given the function of Example 2.6, assume the following probability distribution:

p:	(0, 0, 0) → 0.10	(1, 0, 0) → 0.05
	(0, 0, 1) → 0.15	(1, 0, 1) → 0.05
	(0, 1, 0) → 0.05	(1, 1, 0) → 0.25
	(0, 1, 1) → 0.20	(1, 1, 1) → 0.15

Figure 2.4 shows the two trees of Figure 2.1 (page 18) with their node probabilities; the expected testing cost of tree (a) is

$$E_{(a)} = (0.25 + 0.25) \cdot (5 + 2) + 0.3 \cdot (5 + 6) + (0.05 + 0.15) \cdot (5 + 6 + 2) = 9.4 ,$$

while that of tree (b) is

$$E_{(b)} = 0.35 \cdot 2 + 0.25 \cdot (2 + 5) + (0.25 + 0.15) \cdot (2 + 5 + 6) = 7.65 .$$

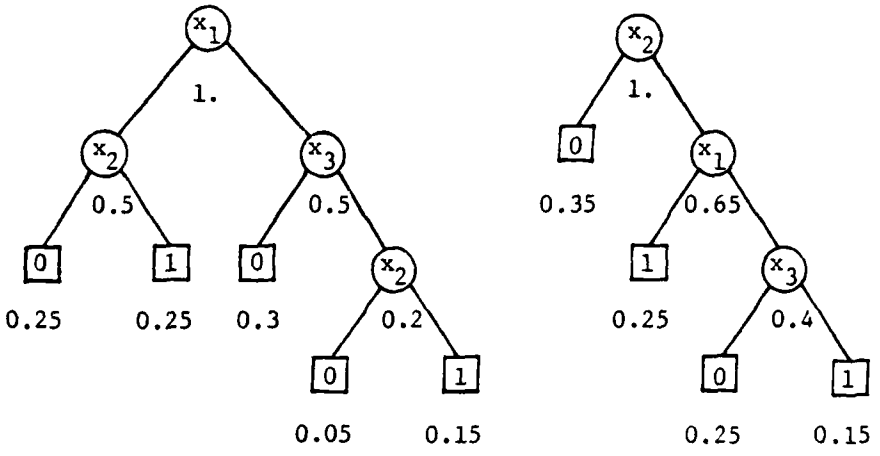


Figure 2.4. The two decision trees of Figure 2.1 with their node probabilities.

The leaf profile of the first tree is (0, 0, 3, 2) and that of the second tree is (0, 1, 1, 2); thus, the second tree has both a larger leaf profile and a smaller reverse leaf profile. □

2.3.2. The general case

Completely specified Boolean functions are only a special case of the system description functions discussed in Chapter 1. In general, the system is described by a (partial) function of discrete-valued variables, $f(x_1, \ldots, x_n)$, where each variable, x_i, can take on exactly m_i values, for some strictly positive integer m_i; without loss of generality, those values will be denoted 0, 1, ..., $m_i - 1$. Those combinations, if any, for which no mapping is specified may, by definition, be assigned any value in the range of the function. (In the case of Boolean functions, such combinations are often termed "don't care entries" and assigned the symbolic value ϕ.)

In the obvious way, $f_{(x_i=k)}$ will denote $f(x_1, \ldots, x_{i-1}, k, x_{i+1}, \ldots, x_n)$, $0 \leq k < m_i$; a variable, x_i, will be deemed redundant if

$$f_{(x_i=0)} = f_{(x_i=1)} = \cdots = f_{(x_i=m_i=1)} ,$$

where $f(x_1', \ldots, x_n') = f(x_1'', \ldots, x_n'')$ if either both combinations are mapped to equal values or at least one of the two combinations has no specified image. A function will be called constant if all combinations of values for which a mapping is specified are mapped to the same value; it is noted that a function, all variables of which are redundant, need not be constant.

The following is the natural extension of Definition 2.1. To the author's knowledge, it is the first formal definition proposed for decision trees.

Definition 2.3. Let $f(x_1, \ldots, x_n)$ be as above. If f is a constant, then the decision tree for f consists of a single vertex labelled by that constant. Otherwise, for each x_i, $1 \leq i \leq n$, f has a decision tree composed of a root labelled x_i and m_i decision subtrees, corresponding to the subfunctions $f_{(x_i=0)}, \ldots, f_{(x_i=m_i-1)}$, in that order. □

The definition of decision diagrams is similarly extended.

As before, storage and testing costs may be specified as well as a probability distribution on the combinations of variables' values. Unless all variables are m-valued for some fixed m (in which case the decision tree is a full m-ary tree), they will likely have different costs; thus

the concept of leaf profile is less useful for general functions than for m-ary (in particular, Boolean) functions. However, all of the other measures defined in Section 2.3.1 are directly applicable to the general case.

Example 2.10. Let f be a partial function of three variables, $f: \{0, 1\}^2 \times \{0, 1, 2\} \rightarrow \{a, b, c\}$, given by the following mapping:

f:	(0, 0, 0) → a	(1, 0, 0) → a	(2, 0, 1) → b
	(0, 0, 1) → a	(1, 0, 1) → b	(2, 1, 0) → a
	(0, 1, 1) → c	(1, 1, 0) → c	(2, 1, 1) → b

Let the variables' costs be as follows:

storage costs	x_1: 1	x_2: 2	x_3: 3
testing costs	x_1: 5	x_2: 3	x_3: 2

Finally, let the probability distribution, p, be given by:

p:	(0, 0, 0) → 0.00	(1, 0, 0) → 0.10	(2, 0, 0) → 0.00
	(0, 0, 1) → 0.05	(1, 0, 1) → 0.05	(2, 0, 1) → 0.20
	(0, 1, 0) → 0.00	(1, 1, 0) → 0.10	(2, 1, 0) → 0.10
	(0, 1, 1) → 0.10	(1, 1, 1) → 0.15	(2, 1, 1) → 0.15

Three of the combinations are assigned a probability of zero; this, however, does not imply that those events are impossible, but merely that they are extremely rare. (This makes provision for the fact that probability estimates must suffer from inaccuracies.) An impossible

event (one that would result in a contradiction) will have a zero probability and no specified mapping; for this function, (0, 1, 0) and (2, 0, 0) can be considered impossible events, while (0, 0, 0) is merely rare. Figure 2.5 shows two decision trees (with leaf probabilities) and associated decision diagrams for the function. The various measures defined in Section 2.3.1 are tabulated below.

measure	α	β	h	η	H	E
tree and diagram (a)	13	10	10	72	9	8.8
tree and diagram (b)	9	9	10	69	8.625	7.85

□

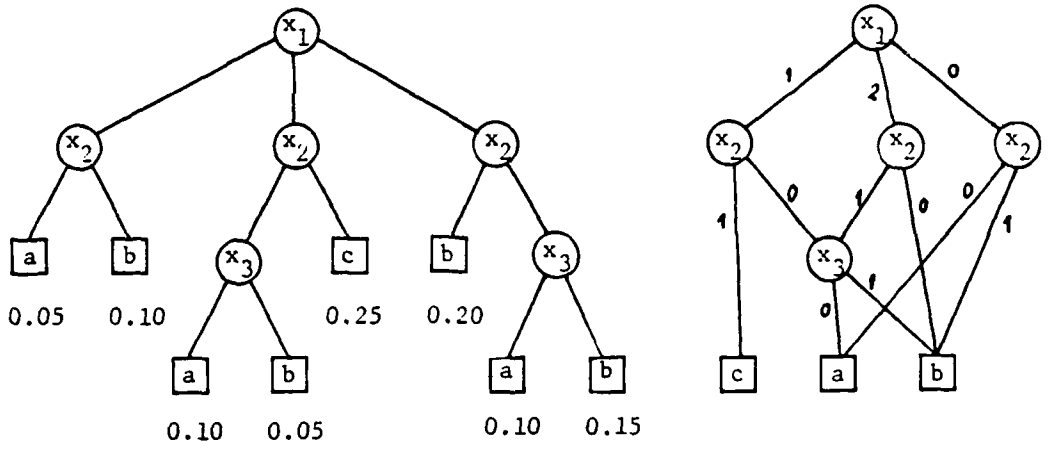
It is noted that a decision tree will, in general, make arbitrary assignments of values to some of the combinations for which no mapping was specified; in fact, it is always possible to find a decision tree that leaves no unspecified entry for functions of binary variables.

Further generalizations to relations, recursive relations, and tree hierarchies will be considered in Chapter 5.

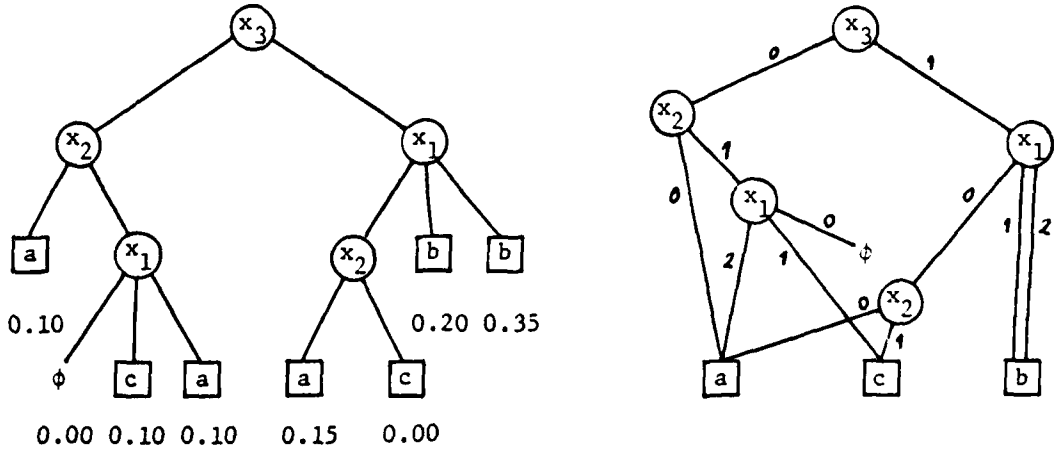
2.3.3. Decision tables

The terminology used for decision tables in the following is that of (Metzner 77).

A decision table is an organizational or programming tool. It can be viewed as a matrix where the upper rows specify sets of conditions and the bottom ones sets of actions to be taken when the corresponding conditions are satisfied; thus, each column, called a rule, describes a procedure of the type "if conditions, then actions " Usually, each condition and action is used as a label on the appropriate row and a rule



(a)



(b)

Figure 2.5. The two decision trees and associated decision diagrams of Example 2.10.

is specified by entering values in the condition rows (or blanks, for irrelevant conditions, called "don't care") and check marks (meaning "perform") in the action rows.

Example 2.11. The following decision table describes how to spend a Saturday afternoon in spring. It has two condition rows, three action rows, and four rules; the first condition is a binary variable (taking values from the set {yes, no}), while the second is a ternary variable (taking values from the set {calm, breezy, windy}).

Raining?	yes	no	no	
Wind condition		breezy	calm	windy
Clean basement	✓			✓
Spade garden			✓	
Fly kites with children		✓		

The four rules can be read as:

"if it is raining, then clean the basement";

"if it is breezy and not raining, then fly kites with the children";

"if it is calm and not raining, then spade the garden";

"if it is windy, then clean the basement." □

A pair of rules overlaps if a combination of condition values can be found that satisfies the condition sets of both rules. If two overlapping rules specify different actions, they are called inconsistent and their table is said to be ambiguous; if they specify identical

actions, they are termed redundant. The order in which rules appear in the table is normally irrelevant. An exception to that case is the so-called else rule, which always appears in the last column of the table; such a rule has no condition entries and is to be used when no other rule in the table can be applied.

Example 2.12. In the decision table of Example 2.11, rules 1 and 4 overlap because they are both applicable when it is raining and windy. Since they specify the same action set, they are redundant, and since no other rules overlap, the table is unambiguous. That same table can be rewritten using an else rule, thereby considerably simplifying it, as shown below.

Raining?	no	no	
Wind condition	breezy	calm	
Clean basement			✓
Spade garden		✓	
Fly kites with children	✓		

□

Tables with an else rule are examples of complete tables, which have an applicable rule for every combination of conditions.

Decision tables described so far are in so-called extended-entry form. Sometimes, however, it is required that all conditions be Boolean variables; this gives rise to limited-entry decision tables. Although most such tables are set up in limited format from their conception, it

may be necessary to convert extended-entry tables to limited-entry format; this is often done by using one Boolean variable for each value (but one) of the multivalued variable to be replaced (Press 65). This process results in tables where entries in one condition row often imply (absent) entries in others; such implied entries can also be present in any decision table and give rise to apparent (but inexistent) ambiguity. Since the implications result from purely semantic considerations, they cannot be detected by an automatic processor; thus, it is imperative that they be specified whenever the table must be logically checked or translated. The impossible combinations of conditions will then be treated (even in tables with an else rule) as inputs with unspecified mapping.

Example 2.13. The decision table of Example 2.12, converted to limited entry, is shown below (with an else rule); it still has three action rows and three rules, but now has three condition rows.

Implied entries are shown in parentheses; their absense, while not confusing to a human, would induce an automatic processor to decide that the first two rules are inconsistent, since both could apparently apply when it is not raining and is calm and breezy. The contradiction inherent in the last two conditions is of semantic origin and thus undetectable by a machine. It is noted, however, that the specification of implied entries in the above table is insufficient: while it

Raining?	no	no	
Calm?	(no)	yes	
Breezy?	yes	(no)	
Clean basement			✓
Spade garden		✓	
Fly kites with children	✓		

identifies the impossible combination (no, yes, yes), it fails to identify the equally impossible combination (yes, yes, yes), which will be erroneously included in the else rule. This suggests that logical inconsistencies be separately listed; for instance, the above table would be supplemented by the logical expression NOT (breezy = yes AND calm = yes). □

Further information about the table is often provided in the form of implementation and testing costs for the conditions and a probability distribution on the rules; when all rules are simple (that is, each applies to a single combination of conditions) and the table is complete, this distribution is equivalent to one specified on the combinations of conditions.

It should now be clear that an unambiguous extended-entry decision table is a special case of a partial function of multivalued variables, where the conditions correspond to the variables and the sets of actions to the function values. In particular, a complete decision table corresponds to a completely specified function, and a limited-entry

decision table corresponds to a function of binary variables. An ambiguous decision table can be assimilated to a relation, a case discussed in Chapter 5. A decision tree representation of a decision table, usually called a sequential testing procedure, is then of particular importance, as it corresponds to an implementation, usually in software, of the decision table. Indeed, the importance of the limited-entry format is in good part due to the ease of programming binary decisions (by if-then-else constructs).

2.3.4. Binary identification problems

Identification is, of course, a fundamental problem in many human endeavors. Of particular interest is the situation where an unknown event or specimen is to be classified into one of a finite number of categories, based upon the outcome of a number of tests. [This is a special case of the concept of questionnaire (Picard 72).] Such problems arise in biology, medical diagnosis, machine troubleshooting, and numerous pattern recognition applications. A binary identification problem includes only binary tests.

Formally, a binary identification problem [as defined in (Garey 72)] consists of:

- a finite set of objects (or categories), $\{O_1, \ldots, O_n\}$, which represents the universe of possible identifications;
- an optional probability distribution function on that set of objects (if absent, the distribution is taken to be uniform);

- a finite set of tests (or questions), $\{Q_1, \ldots, Q_m\}$, each of which is a subset of the set of objects (thereby listing the possible identifications for the unknown object as determined by a positive answer to that particular test);
- optional sets of storage and testing costs associated with the set of questions (when unspecified, costs are taken to be unity).

In most cases, the size of the set of objects, n, is larger than the size of the set of questions, m. A solution to such a problem is an identification procedure, that is, a decision tree where internal nodes are associated with questions and leaves with objects.

Example 2.14. Let a binary identification problem be given by a set of four objects, $\{O_1, O_2, O_3, O_4\}$, with respective probabilities, $\{0.1, 0.2, 0.3, 0.4\}$, and a set of three tests, $\{Q_1, Q_2, Q_3\}$, with $Q_1 = \{O_1, O_2\}$, $Q_2 = \{O_1, O_4\}$, $Q_3 = \{O_4\}$, and unity costs. Two identification procedures for this problem are shown in Figure 2.6. □

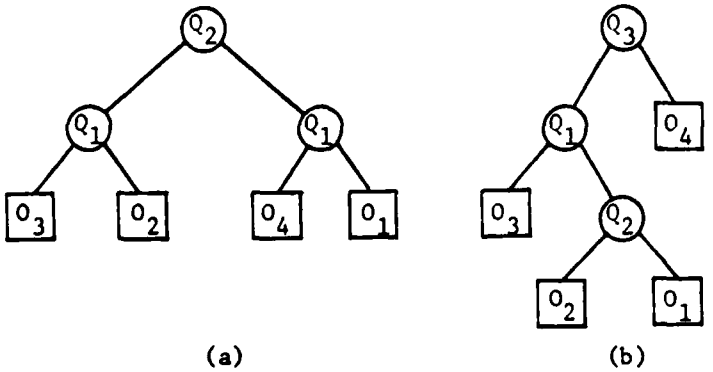


Figure 2.6. The two identification procedures of Example 2.14.

An identification problem is clearly a special case of a partial function of binary variables, where the questions correspond to the variables and the objects to the function values. In fact, a binary identification problem corresponds to an injective and surjective partial function, since exactly one combination of variables' values is mapped to each object.

As a result, decision trees for binary identification problems have a fixed number of leaves (one per object) and thus of internal nodes (since the number of internal nodes of a full binary tree is one less than the number of its leaves). Since no two leaves are identical, there can be no common subtrees, so that decision diagrams for identification problems are decision trees. Another consequence is that, when storage costs are unity, all decision trees for the problem have the same tree (and, of course, diagram) storage cost (the number of objects minus one).

Example 2.15. The various tree and diagram measures defined in Section 2.3.1 are tabulated below for the two trees of Figure 2.6.

measure	α	β	h	η	H	E
tree (a)	3	3	2	8	2	2
tree (b)	3	3	3	9	2.25	1.9

Thus, the second tree has a lesser expected testing cost than the first, even though its height is greater and its root is associated with the redundant variable, x_3 (which the first tree does not test at all). It is noted that H is in constant ratio to η, since the number of leaves

is constant. If it is now assumed that the probability distribution is uniform, then E becomes equal to H, so that only three essentially different measures are left (not counting the profiles): α, h, and E. □

CHAPTER 3

SURVEY OF PREVIOUS WORK

3.1. Introduction

The choice of decision trees as models of functions of discrete variables raises several questions. First, does this model capture important aspects of functions that are not reflected in other models (such as Boolean calculus for logic functions)? Next, since several measures have been used on decision trees, what can be said of those measures when applied to functions, and how can optimal tree representations be developed? Finally, is it possible to develop from the model a measure of complexity (as independent of the model as possible) and to apply it to practical problems?

Since decision trees can be regarded as sequential evaluation procedures, most of the published results about them concern the conversion of parallel data to optimal and suboptimal decision trees. Three principal lines of investigation have progressed independently so that efforts have often been duplicated. In the following sections, each area will be reviewed separately; however, connections will be explicitly mentioned and all cases will be expressed in the general framework introduced in Section 2.3.

The problem of converting a discrete function to an optimal decision tree is a difficult one and its exact complexity remained unknown for years; therefore, both optimal and heuristic, suboptimal algorithms

abound. Only the most significant will be reviewed here; as will be seen, they are representative of a very few basic techniques which, with slight variations, comprise almost all of the proposed algorithms. The case of binary identification, as found in taxonomy and machine diagnosis, is first reviewed, since it constitutes a rather restricted subproblem and also because it was the first to be investigated. [Indeed, the use of decision trees in biology--where they are called diagnostic keys--is thought to go back to Aristotle and Theophrastus (Morse 71).] The conversion of decision tables to computer programs, which has engendered a wealth of articles, is surveyed next. The more general problem of representing discrete functions by decision trees and diagrams, which has been studied in the context of switching theory, pattern recognition, and concrete complexity theory, is then reviewed. The various findings are regrouped in a short summary of the "state-of-the-art" knowledge about decision trees.

3.2. Diagnosis and Identification

The simplest form of sequential evaluation is a linear sequence where, at each step, only one path leads to another test. This corresponds to a degenerate tree with a number of internal nodes equal to its height. Finding an optimal tree then reduces to choosing one of the test sequence permutations; moreover, tne only applicable criterion is the expected testing cost. Variants of this problem were studied by (Johnson 56, Hoehn 58, Riesel 63, Slagle 64, Hanani 77), who gave simple necessary and sufficient conditions for an ordering (in terms of the

ratio of cost to probability) that minimizes the expected testing cost (or time).

In a typical application to machine diagnosis or specimen identification, many more variables are considered than are needed to distinguish all values. Thus, it makes sense to attempt to minimize the size of the set of variables (or its cost if sensor costs--as defined in Section 2.3.1--have been specified). This corresponds to finding a tree of minimum height when the order of testing is constrained to be the same on all branches of the tree; the corresponding decision problem was recently proved to be NP-complete (Garey 79, p. 222). [Thus, the simple exhaustive search method proposed in (Willcox 72) is not much worse than what might reasonably be expected.] The complexity of an optimal algorithm was recognized early and a heuristic selection criterion developed (Gyllenberg 63, Chang 65, Chang 70). According to this criterion, the first variable selected is that which gives rise to the largest number of pairs of values; since this happens when the set of values is divided into subsets of most nearly equal sizes, this is an example of a splitting algorithm. Once a variable is selected, the same computations are carried out independently on the resulting subsets and the results added to determine the next variable to be selected. The criterion was enlarged in (Chang 65) to include the case of arbitrary partial functions of binary variables; in this case, only the pairs of distinct values are tallied. No results were published on the performance of this selection algorithm.

It is noted that this criterion is easily extended to take into account sensor costs by using the ratio of the number of pairs of (distinct) values to the sensor cost as the critical measure.

The problem of constructing optimal binary identification trees was first seriously considered in (Brulé 60, Kletsky 60). In the first paper, the number of different decision trees (called sequential test diagrams) for n objects with all $2^n - 1$ binary tests available was estimated to be of the order of $1.78^n \cdot n!$ and that of distinct leaf profiles with n leaves of the order of 1.84^n (although the concept of leaf profile was not mentioned). The authors also defined the expected and worst case testing costs, E and h; in particular, they showed that when all $2^n - 1$ tests are available and costs are unity, the tree corresponding to the Huffman code (Huffman 52) minimizes the expected testing cost [a result previously derived in (Zimmerman 59)].

The analogy with coding and information theory was further pursued in the second paper, where the entropy of a function was defined. Expressed in the notation developed in Section 2.3, the entropy of any partial function of n variables, $f(x_1, \ldots, x_n)$, is the quantity:

$$H(f) = - \sum_{v} p(f = v) \cdot \log_2 p(f = v) \ ,$$

where $p(f = v)$ is the probability that f takes the value v (i.e., the sum of the probabilities of all n-tuples, $(x_1, \ldots, x_n)$, that are mapped to v) and the sum is taken over all values, v, in the range of f. The entropy of a function can be considered as expressing its initial

ambiguity or its information content [as in (Rescigno 61), where it is called "repartment"]. The authors then proposed to rank the variables of a function by the ratio of the ambiguity they remove to their cost, where the ambiguity removed by a variable, x_i, which takes on m_i values, can be expressed as:

$$H(x_i) = - \sum_{j=0}^{m_i-1} p(x_i = j) \cdot \log_2 p(x_i = j) \ .$$

Thus, the ambiguity removed by a variable also has the form of an entropy; indeed, it was independently derived as an entropy function in (Mandelbaum 64).

Using the removed ambiguity per unit cost as a selection criterion in constructing decision trees yields the information algorithm, of which a large number of published heuristics is a special case. In particular, when costs are unity and the probability distribution is uniform, the information algorithm chooses that variable which partitions the image of the function into the most nearly equal subsets; this is recognized as a special case of the splitting algorithm. Numerous publications make use of one or the other of these algorithms, with appropriate modifications, to solve problems in machine diagnosis and biological classifications (La Macchia 62, Winston 69, Pankhurst 70, Morse 71, Gower 72). However, no analysis of the criteria was published until 1974, when (Garey 74) showed that the ratio of the expected testing cost of a tree constructed by the splitting algorithm to that of an optimal tree could be arbitrarily large, even in the case of uniformly

distributed objects. [For n objects the ratio can be as large as $\log_2 n / \log_2 \log_2 n$; if arbitrary probability distributions are allowed, the ratio can be at least n (Garey 80)]. This disproved a longstanding conviction that the splitting algorithm was optimal for uniform distributions, as quoted in (Brulé 60, Kletsky 60, Osborne 63). However, (Hung 74) showed that the splitting algorithm is asymptotically optimal, in the sense that, as the number of variables grows, the expected value of the ratio of the cost of the trees constructed by the splitting algorithm to that of the optimal trees converges to one.

Algorithms for constructing a decision tree with minimal expected testing cost were presented in (Garey 72, Misra 72) (the first for binary tests, the second for multivalued tests). Both algorithms make use of dynamic programming and may require time exponential in the size of the input (in contrast to the quasi-linear splitting and information algorithms). There is, however, little likelihood of improvement since the decision problem associated with the construction of optimal decision trees for partial bijective functions is now known to be NP-complete (Hyafil 76, Loveland 79).

3.3. Decision Tables

The conversion of decision tables to computer programs using decision trees has been the subject of numerous articles over the past fifteen years. However, a majority of these articles repeat the results presented in others or contain erroneous statements (often masked by a different--and sometimes cryptic--notation). Consequently, only those articles of

actual or historical importance will be reviewed. Further references are available in the book by (Metzner 77), the survey paper of (Pooch 74), or the special issue of SIGPLAN Notices (September 71).

The subject was first studied in (Montalbano 62). Two desirable objectives are mentioned, namely minimizing storage cost and expected testing cost; to that effect, two heuristic selection strategies are presented. The first strategy selects variables which tend to maximize the profile of the tree by reaching leaves as soon as possible; this is claimed (erroneously, as will be seen in Chapter 4) to minimize storage cost. The second rule (called "delayed rule") is intended to minimize the expected testing cost by selecting at each step that variable which divides the decision table into most nearly equal subtables; hence, it is a special case of the splitting algorithm. This rule was refined in (Pollack 65) to take into account the fact that several tables may represent the same decision process; it was proposed to minimize the number of rules leading to the same set of actions by the Quine-McCluskey algorithm, thereby yielding a minimal equivalent table (which, however, is still not unique). The author overlooked the fact that the Quine-McCluskey algorithm requires exponential time [in fact, the minimization of a limited-entry decision table or of a Boolean formula is an NP-hard problem (Masek 80)], so that his suboptimal algorithm requires exponential time, just like an exhaustive search for the optimal tree. The same oversight can be found in several otherwise important articles (Ganapathy 73, Shwayder 75).

Pollack's algorithm for minimizing the expected testing cost consists of several rules of thumb, which are roughly equivalent to the information algorithm. This was recognized by (Shwayder 71), who proposed entropy as a selection criterion for limited-entry decision tables with one rule per action set; the full information algorithm was presented in (Ganapathy 73, Shwayder 74). Unfortunately, both authors use the decision table rather than its underlying function as a basis for the algorithm; in consequence, there are problems of nonuniqueness of representation and of table minimization.

Further heuristic selection rules for minimizing the expected testing cost were presented by (Verhelst 72, Sethi 80). The first author based his criterion on a lower bound estimate, later shown to be incorrect (King 74), while the second author used a one-step look ahead with what amounts to an information algorithm.

Montalbano's "quick rule" for minimizing storage cost was also successively refined by several researchers. In (Rabin 71), it is proposed to select that variable which results in a minimum number of rules being split; this requires that a minimal disjoint table be first obtained. The idea is extended to multivalued conditions in (Michalski 78), and refined by considering the minimum number of disjoint rules necessary in the original table and that necessary for the sub-tables determined by the variable under consideration. Neither author discussed the problem of generating an equivalent table with a minimum number of disjoint rules; this is again a fatal flaw, since that problem is itself NP-hard. A similar analysis was performed in (Yasui 71,

Yasui 72) for limited-entry tables; it was showed that, when applied to limited-entry tables with n conditions and unity storage costs, this selection algorithm (called iterated local minimization) may construct a tree with at least 2^{n-4} more nodes than the optimal tree; a comparison with Montalbano's original strategy showed that each algorithm can construct trees with 2^{n-4} more nodes than those constructed by the other.

The first thorough analysis of the conversion of limited-entry decision tables to decision trees and diagrams was a two part article by (Reinwald 66, Reinwald 67), whose excellent work was unfortunately illserved by an exceedingly complex notation. In the first part, the authors derived a lower bound on the expected testing cost of partial decision trees as a function of the variables already tested (which will be further examined in Chapter 5) and used it as the basis for a fast suboptimal strategy (of local optimization) and for an optimal seeking branch-and-bound procedure (which, unfortunately, may require exponential time). In the second part, a simple lower bound on the storage cost of a partial decision tree was derived in terms of the irredundant conditions and used again for a fast local optimization procedure and a (sometimes exponential) branch-and-bound algorithm; the authors further showed how to modify the branch-and-bound criterion to obtain decision diagrams with minimum storage cost. This latter procedure remains to this date the only one applicable to decision diagrams. It is noted that, in both papers, the authors used a table where all rules are simple, which is equivalent to giving the mapping of the underlying function. This guarantees that the table has a unique representation and avoids the need for minimization.

The problem of converting limited-entry decision tables to optimal decision trees finally received an efficient solution in 1973 with the publication of a dynamic programming algorithm (Bayes 73). This solution was independently rediscovered by (Schumacher 76) who generalized it to extended-entry tables; a somewhat faster version (due to the use of added heuristics inspired from game trees) was recently published (Martelli 78). As is the case for dynamic programming solutions, the algorithm builds the optimal tree from the leaves up by identifying optimal subtrees; it is thus applicable to any criterion of optimality with the property that an optimal solution must have optimal subsolutions. In particular, it can be used to find the optimal trees with respect to the tree storage cost, the expected and worst case testing costs, and the minimum reverse and maximum profiles, but not with respect to the diagram storage cost nor with the total and normalized testing costs. The running time of the algorithm is of the order of the number of possible nodes that are examined, where each node corresponds to a combination of tested and untested variables. For a function of n k-ary variables, there are $(k + 1)^n$ distinct nodes and generating them from the bottom k^n nodes (all variables tested) to the top node (all variables untested) requires a number of steps equal to:

$$\sum_{i=1}^{n} i \cdot \binom{n}{i} \cdot k^{n-i} = n \cdot (k + 1)^{n-1} ,$$

since a node with i untested variables has i possible parents (each with one more untested variable) and can be chosen in $\binom{n}{i} \cdot k^{n-i}$ ways. Since

the input is of size $I = \mathcal{O}(k^n)$, the dynamic programming algorithm requires $\mathcal{O}\left(I^{\log_k(k+1)} \cdot \log I\right)$ and is thus very efficient. (For binary variables, the complexity is $\mathcal{O}(I^{1.585} \cdot \log I)$; when k becomes very large, the complexity converges to $\mathcal{O}(I \cdot \log I)$.)

None of the heuristic or optimal algorithms mentioned above is directly applicable to ambiguous decision tables. The problems posed by these tables were analyzed in (King 73), who concluded that a pair of inconsistent rules could be taken to mean that both action sets should be applied, or either one, in an arbitrary manner. As will be seen in Chapter 5, this behavior can be characterized by using relations instead of functions in the description of decision tables.

3.4. Representation and Evaluation of Functions

Decision trees can be used as representations of functions for purposes of software or hardware implementation, or as an analytical tool. Some of those applications will now be reviewed.

Since a k-ary decision tree is the exact equivalent of a hardware multiplexer tree, it is possible to synthesize any function of k-ary variables with only one type of element--a k-to-one multiplexer; the resulting circuit will have few interconnections and will lend itself well to large scale integration. Moreover, some multiplexers may feed several others (have more than one "parent") to form a multiplexer network, the equivalent of a decision diagram. This has attracted researchers in switching theory, who studied the problem of minimizing the number of multiplexers (that is, the number of nodes in the tree) or

the worst case propagation delay (that is, the height of the tree). An algorithm to minimize the size of a multiplexer tree was presented in (Mange 78, Cerny 79); however, it necessitates the generation of all implicants of the function as well as of their subsequent combinations in order to find the optimal solution and thus requires exponential time. The same idea appeared in (Davio 77) for the minimization of the height of the tree. Both approaches were regrouped and generalized to k-ary functions in (Thayse 78).

Since decision trees model multistage decision processes, they have found widespread applications in sequential pattern recognition. Although a tree is often synthesized directly from the problem without attempt to optimize its cost (Rounds 79, You 76), there have been a few studies of the optimization problem. An approach using game tree searching techniques was proposed in (Slagle 71), who noted that this approach can be modified to yield optimal dynamic programming or branch-and-bound algorithms. The heuristic procedure presented in (Sethi 80) for decision tables had first been designed for pattern recognition purposes (Sethi 77). More importantly, a dynamic programming algorithm similar to those of (Bayes 73, Schumacher 76) was published by (Meisel 73) [and later refined by (Payne 77)]. This algorithm takes a multivalued function as input and produces a binary decision tree optimal with respect to expected or worst case testing cost, or storage cost, or any weighted combination of those costs.

The evaluation of Boolean functions using decision trees with an eye toward software implementation has given rise to several optimal and

heuristic algorithms. A selection criteria similar to that used in (Cerny 79) was proposed in (Halpern 74a, b); it necessitates the generation of all prime implicants for the function and its dual (hence requiring exponential time), implicants which are then ranked in terms of their probability to cost ratio (where the cost of an implicant is that of the optimal tree for it). Variables which appear in both the best implicant for the function and that for its dual are then selected. The author shows that this strategy is optimal for symmetric functions (those that remain invariant under any permutation of the variables), but offers no analysis of performance in the general case. (Breitbart 75a) presented a heuristic selection rule for monotone Boolean functions with unity costs and uniform probability distribution, which requires to find the minimal sum-of-products form (known to be unique for monotone functions) and is thus exponential time; in a later analysis (Breitbart 78), it was shown that trees constructed by this rule can have an expected number of tests at least (n/log n) times larger than the optimal trees for functions of n variables. The same authors also adapted the work of (Reinwald 66) to Boolean functions (Breitbart 75b); their article contains a theorem relating Chow parameters to the expected testing cost of a decision tree (which will be further examined in Chapter 5) and provides most of Reinwald and Soland's results in a much more readable form. In the special case where the function is composed of a disjunction of disjoint functions, (Perl 76) showed that testing all functions in some fixed order is no worse than changing the order of testing on some paths, thereby generalizing some results of (Slagle 64).

The worst case number of tests (the height of a tree) indicates a minimum number of operations that must be performed in order to compute a function; as such, it is a useful technique for deriving lower bounds to be used in concrete complexity theory. Of particular interest is the worst case behavior of Boolean functions. A function is said to be exhaustive if every tree for the function has a path on which all variables are tested; (Rivest 76a, b) proved that almost all (in the sense of asymptotics) Boolean functions are exhaustive. (The proof will be presented in Section 4.3.3 along with some new results on exhaustiveness.)

Lower bounds are also helpful if one is to compare decision trees with other representations of functions. In a fundamental paper, (Lee 59) introduced binary decision diagrams (which he called binary decision programs) and compared them to Boolean formulae as means of representation of Boolean functions. Using an ingenious proof technique, he was able to show that no Boolean function of n variables needs more than $4 \cdot 2^n/n - 1$ diagram nodes and that some require more than $1/2 \cdot 2^n/n$ nodes. (A straightforward extension of his proof shows that the number of diagram nodes needed to represent any k-ary function of n variables is bounded below by k^n/kn and above by $2 \cdot k^n/n$, for $k > 2$.) By contrast, it is well-known (Savage 76) that a Boolean formula may require up to $\mathcal{O}(2^n/\log n)$ operators for a function of n variables; moreover, every operation must be carried out at each evaluation of the function, so that the expected number of operations needed to evaluate a Boolean formula is $\mathcal{O}(2^n/\log n)$ while a decision diagram will never

need more than n comparisons. The author justly concluded that binary decision diagrams should be further investigated as efficient representations of Boolean functions. Some of the interesting properties of decision diagrams and trees were independently rediscovered by (Prather 78) (who called them atomic digraphs) and (Akers 78a, b), who investigated their use in testing switching circuits. (This will be further pursued in Chapter 6.)

Finally, a decision tree can be used to model the control structure of a program (Prather 78), in particular in relation with Ianov's schemata (Ianov 60). It then becomes important to recognize identical structures, that is, to decide whether or not two free Boolean graphs are equivalent (i.e., represent the same function). (Fortune 78) showed that this problem is NP-complete; however, (Blum 80) provided a probablistic algorithm that solves the problem in polynomial time.

3.5. Summary

Most of the results available about decision trees as representations of discrete functions concern algorithms for constructing optimal or suboptimal trees. Two types of optimal algorithms have been published. The first uses branch-and-bound techniques and is applicable to all optimality criteria for which some kind of lower bound can be derived; the second uses dynamic programming and is applicable only to criteria that are "compatible" with decomposition in that optimal solutions must have optimal subsolutions. Numerous suboptimal algorithms have been proposed, all of which can result in trees that are arbitrarily far from

the optimal. Most of these rules fall into one of three categories, namely, the splitting strategy, the information algorithm, and the local minimization (using a lower bound).

The problem of constructing optimal decision trees is known to be NP-hard (for most criteria) in the case of binary identification. In the general case, a dynamic programming algorithm is available, which provides an efficient solution (less than $\theta(s^2)$ for input of size s). The three main heuristics are of complexity $\theta(s \cdot \log s)$ or $\theta(s^2)$, but several others have exponential complexity due to the incorporation of an NP-hard problem (such as minimum cover or minimum sum-of-products form).

Much less attention has been devoted to finding general characteristics of tree and diagram representations of functions. It is known, however, that almost all Boolean functions are exhaustive, that is, have tree representations of maximal height. Further, it has been shown that at most $\theta(2^n/n)$ nodes are needed in a decision diagram to represent any Boolean function of n variables (versus $\theta(2^n/\log n)$ operators in a Boolean formula), which can then be evaluated in $\theta(n)$ comparisons (versus $\theta(2^n/\log n)$ operations with a Boolean formula).

Most of the work done has been in relation with Boolean functions and it must be noted that an important subset of the results do not generalize to multivalued functions.

CHAPTER 4

MEASURES AND OPTIMIZATION PROBLEMS

4.1. Introduction

The use of decision trees as models of discrete functions presents a problem of uniqueness of representation, since a given function has, in general, numerous decision tree representations. As indicated in Section 2.3, several criteria have been proposed to select a standard (optimal) representation; however, the construction of this standard tree may be quite a complex problem for several criteria. Moreover, there is a fundamental question of choice, since at least seven criteria have been defined.

The complexity of the optimization problem for the diverse criteria and types of functions has received rather limited attention, as mentioned in Chapter 3. The decision problem for each of the seven criteria is easily seen to be in NP. The exact complexity of the decision and optimization problems will be discussed in this chapter.

The choice of a criterion has rarely been discussed in the literature. Most researchers used the expected testing cost or the storage cost as corresponding to demands on time and memory, respectively, in software implementations. (Verhelst 72) argued that the storage cost is rather unimportant, but did not further justify his use of the expected testing cost. The question was avoided in several articles (Yasui 71, Payne 77) by proposing to find a tree representation that would simultaneously satisfy a few criteria. This approach, however, is

inapplicable since, as will be shown in the following sections, almost all the proposed criteria are incompatible in the sense that they cannot be optimized simultaneously. This chapter investigates the relationships between the various measures and discusses the merits of each; it shows that only one criterion satisfactorily reflects the complexity of decision trees.

4.2. The Case of Binary Identification

The various applicable measures will be first examined under the assumption of unity costs and uniform distribution of objects. Under those conditions, the storage cost of a tree reduces to its number of nodes (which is fixed, as noted in Section 2.3.4), and its expected testing cost is equal to its normalized testing cost, of which the path length is a fixed multiple; hence, only four criteria are applicable, namely, the height, the path length, and the minimum reverse and maximum profiles.

Proposition 4.1. The maximum profile is incompatible with the other three measures.

Proof: Consider the identification problem with five objects, $\{a, b, c, d, e\}$, and four tests, $\{T_1 = \{a\}, T_2 = \{a, b\}, T_3 = \{a, b, c\}, T_4 = \{a, b, c, d\}\}$. The trees with maximum profile test T_1 or T_4 first, while those optimal with respect to the other three criteria test T_2 or T_3 first, with the resulting measures listed below.

first test	leaf profile	height	path length
T_1 or T_4	(0, 1, 1, 1, 2)	4	14
T_2 or T_3	(0, 0, 3, 2, 0)	3	12

□

It is further noted that the function used in the proof also has trees rooted in T_1 or T_4 with a leaf profile of (0, 1, 0, 4, 0) and thus a height of three and a path length of thirteen; hence, minimizing the height of a tree does not result in the optimization of any other measure. (On the other hand, it is obvious that minimizing the reverse profile will minimize the height.) The known relationships between the four measures are summarized in Figure 4.1, where the empty set symbol, ϕ, means that the corresponding measures are incompatible (that is, that the set of all trees optimal under one criterion has no intersection with that of all trees optimal under the other criterion). The exact relationship between the reverse profile and the path length is unknown. It is easy to construct an example showing that they are not strictly equivalent, in the sense that the set of all trees for the example does not get ranked in the same order by both criteria; however, this author was not able to construct a problem where the optimal trees for both criteria did not coincide.

The introduction of nonuniform probabilities results in further incompatibilities and one more measure (the expected testing cost); in fact, the only two measures that are not incompatible are, trivially, the reverse profile and the height. Storage and testing costs invalidate the use of leaf profiles, but give rise to another valid measure (the tree storage cost); all measures are then pairwise incompatible.

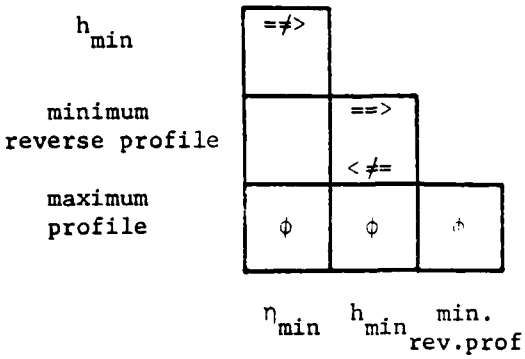


Figure 4.1. Known relationships between the four measures applicable to binary identification trees.

The decision problem for the path length measure was shown to be NP-complete in (Hyafil 76), even with unity costs and uniform distribution. Their construction [a straightforward reduction from the exact cover by three sets--see (Garey 79, p. 53)] can be used to show that the decision problems for the reverse profile and the expected and worst case testing costs are also NP-complete. Using standard completion and search techniques (as developed in Section 2.2.3), it is an easy matter to show that the optimization problems for the storage cost and the total (and thus also normalized), expected, and worst case testing costs are all NP-equivalent. The optimization problem for the reverse profile is not known to be NP-easy: although one can apply the completion technique, a binary search (required since the number of distinct profiles grows exponentially with the number of leaves) would necessitate a polynomial time ranking algorithm for leaf profiles, which is as yet undiscovered. Table 4.1 summarizes the known results about the complexity of decision tree optimization in binary identification problems.

Table 4.1
The Complexity of Optimal Binary Identification Trees

Criterion	α_{min}	π_{min}, H_{min}	E_{min}	h_{min}	rev. prof.	max. prof.
Complexity	NP-equivalent	NP-equivalent	NP-equivalent	NP-equivalent	NP-hard	—

4.3. The Case of Completely Specified Boolean Functions

4.3.1. Relationships between measures: conjectures and counterexamples

The relationships betwen the various measures will be examined in the simplest (and least conducive to incompatibilities) case, where all costs are unity and the probability distribution on the input combinations is uniform. All of the eight measures defined in Section 2.3.1 are then applicable.

Proposition 4.2. The maximum profile is incompatible with any other measure.

Proof: Two counterexamples will be used. First, let f_a be the Boolean function of four variables given by the formula:

$$f_a(x_1, x_2, x_3, x_4) = \overline{x_1}x_3 + x_1x_2x_4 + x_2\overline{x_3}\overline{x_4} .$$

The trees with maximum profile use as first test either x_1 or x_3 and also have minimal expected testing cost; the optimal tree for all other measures, however, tests x_4 first and is unique (except for the minimum

diagram storage cost, which can also be attained by testing x_1 or x_3 first, but with a structure different from that of the maximum profile trees). The various measures for the three types of trees are listed below:

first test	profile	α_{min}	β_{min}	η_{min}	H_{min}	h_{min}	E_{min}
x_4	(0, 0, 0, 8, 0)	7	6	24	$3\overline{}$	3	3
x_1 or x_3	(0, 0, 1, 4, 4)	8	6	30	$3.\overline{3}$	4	3
x_1 or x_3	(0, 0, 2, 2, 4)	7	7	26	3.25	4	2.75

Hence, the maximum profile (and, incidentally, the minimum expected testing cost) is incompatible with the minimum reverse profile, the diagram storage cost, and the total, normalized, and worst case testing costs. Secondly, let f_b be the Boolean function of five variables given by the formula:

$$f_b(x_1, x_2, x_3, x_4, x_5) = x_1x_5 + \overline{x}_1\overline{x}_2\overline{x}_5 + \overline{x}_2x_5 \cdot (\overline{x}_3x_4 + x_3\overline{x}_4) + (x_2 + \overline{x}_5) \cdot (\overline{x}_3\overline{x}_4 + x_3x_4) .$$

The trees with the maximum profile test x_5 first, while those optimal with respect to all other measures first test x_1, with the following results:

first test	profile	α_{min}	β_{min}	η_{min}	H_{min}	h_{min}	E_{min}
x_1	(0, 0, 1, 1, 8, 4)	13	8	57	4.07	5	3.5
x_5	(0, 0, 1, 2, 2, 12)	16	9	76	4.47	5	3.625

Hence the maximum profile is also incompatible with the minimum tree storage and expected testing costs. □

The first function also shows that minimizing the tree or diagram storage costs does not optimize any other criterion, while the second demonstrates the same result about the minimum worst case testing cost.

Proposition 4.3. The normalized testing cost is incompatible with any other measure except, possibly, the worst case testing cost; moreover, minimizing the normalized testing cost may involve the introduction of redundant tests.

Proof: Let f_c be the Boolean function of five variables given by the formula:

$$f_c(x_1, x_2, x_3, x_4, x_5) = x_1x_2 + x_2 \oplus x_3 \oplus x_4 \oplus x_5 ,$$

where $\oplus$ stands for summation modulo 2. The optimal trees for all measures but H test x_1 or x_2 first and use no redundant test, while the trees with minimum normalized testing cost may test any variable first and, in case x_1 or x_2 is chosen, use redundant tests. Two diagrams rooted in x_1 are shown in Figure 4.2, the left being optimal with respect to all criteria but H, for which the right diagram is optimal; the corresponding measures are listed below:

tree	profile	α_{min}	β_{min}	η_{min}	H_{min}	h_{min}	E_{min}
left	(0, 0, 2, 0, 0, 16)	17	8	84	$4.\overline{6}$	5	3.5
right	(0, 0, 0, 4, 0, 16)	19	9	92	4.6	5	4

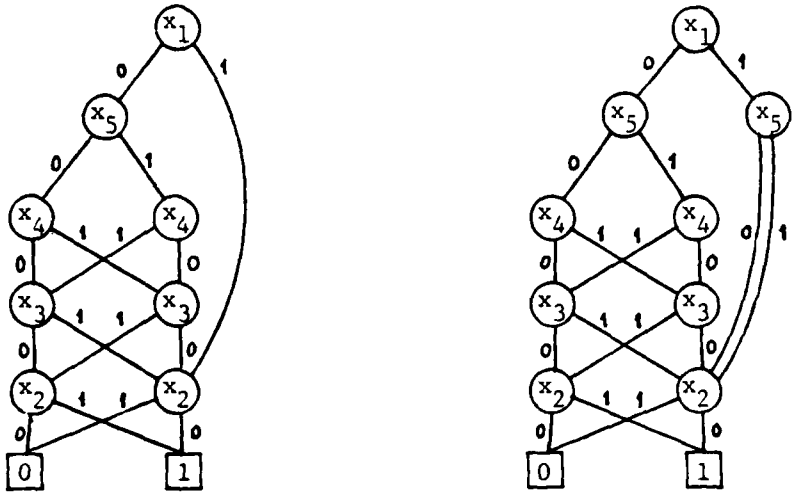


Figure 4.2. The two diagrams for the counterexample of Proposition 4.3.

It is noted that the test of x_5 as the right child of the root is redundant. □

The known relationships between measures are summarized in Figure 4.3 (which incorporates a few results from unmentioned counterexamples). These results disprove several conjectures or assertions found in the literature. In particular, (Yasui 71) claimed that the number of nodes of a tree is a special case of the expected testing cost; this was reiterated as a conjecture in (Breitbart 75a). That it is false can easily be seen by examining the trees for the Boolean function of four variables given by the formula:

$$f(x_1, x_2, x_3, x_4) = x_1x_2 + \overline{x_1}x_3 + x_2\overline{x_3}x_4 \ .$$

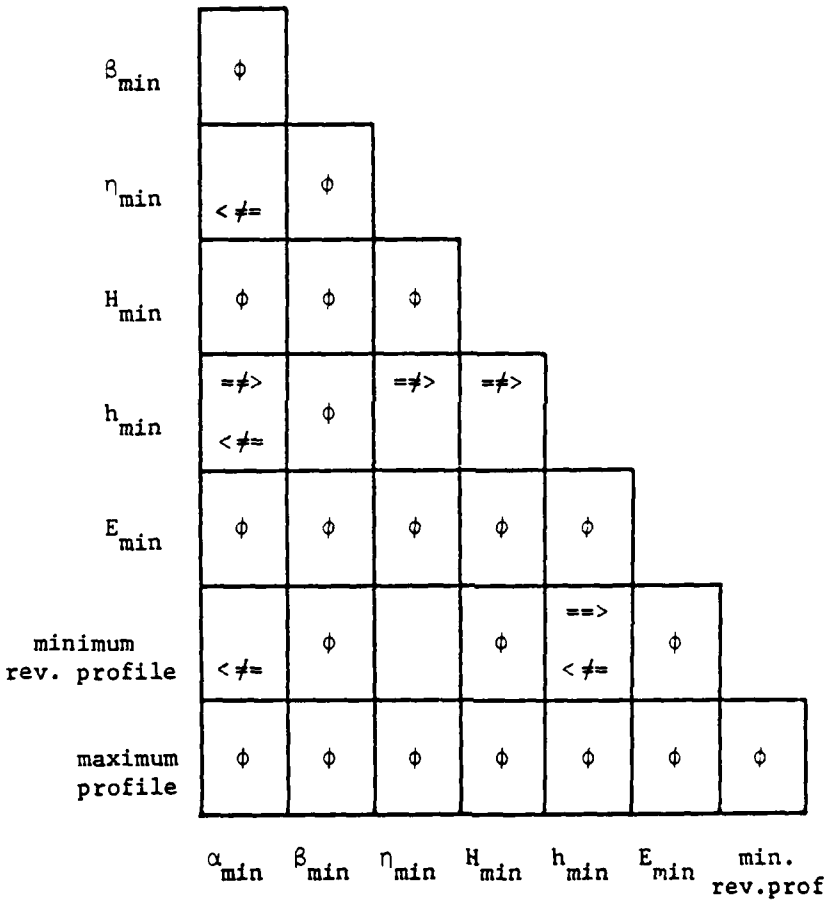


Figure 4.3. Known relationships between the eight measures applicable to binary decision trees.

Entries left blank in the diagram of Figure 4.3 stand for relationships that could neither be proved nor disproved. It is conjectured that these relationships, most particularly the implication (η_{min} ==> α_{min}), hold true. As for identification problems, the introduction of nonuniform probability distributions or nonunity costs renders all measures pair wise incompatible.

4.3.2. Uninteresting measures

From the preceding results, it clearly appears that the normalized testing cost is unsuitable as a measure of complexity on decision trees. Also, since the notion of leaf profile cannot be easily extended to take variables of different costs into account, both the maximum and the minimum reverse profiles are unsuitable. Finally, the tree storage cost is not an accurate measure of the true hardware (or memory) requirements, since the diagram storage cost is never larger and often much smaller. (It is recalled that $\Theta(2^n)$ tree nodes are needed in order to represent any Boolean function of n variables, while $\Theta(2^n/n)$ diagram nodes suffice.) Thus, the tree storage cost is rejected in favor of the diagram storage cost.

Of the four remaining measures, three (η, h, E) are related to problems of tree usage and one (β) to problems of implementation. While the diagram storage cost certainly is the relevant measure of implementation problems (except, perhaps, for the difficulty of its optimization--not known to be polynomially feasible, whereas the tree storage cost can be efficiently optimized), it appears as less

characteristic of the complexity of a function than measures of testing costs. In particular, the storage cost is not as strikingly different for decision trees (as compared with other models of functions) than the testing costs; for instance, although it is an order of magnitude better than Boolean formulae for logic functions--$\Theta(2^n/n)$ versus $\Theta(2^n/\log n)$, this remains small compared to the ratio for the expected testing cost--$\Theta(n)$ versus $\Theta(2^n/\log n)$. Thus, the storage cost does not capture an essentially new aspect of functions, while the expected testing cost definitely does.

Finally, of the three measures of testing cost, only two (h and E) are concerned with the performance of a tree representation. The total testing cost does not make use of the probability distribution, yet neither does it measure a performance extreme (as the worst case testing cost). Although it is of interest in the case of binary identification (since it is then a measure of the cost incurred in identifying one object in each category, that is, in producing each output of the function exactly once), it does not, in general, correspond to practical concerns that an engineer or designer might have about a function.

This leaves two measures of special interest, namely, the expected and worst case testing costs, which are discussed in turn in the next sections.

4.3.3. The worst case testing cost

As mentioned in Section 3.4, (Rivest 76a, b) proved that almost all Boolean functions are exhaustive, that is, have a maximal worst case

testing cost. In this section, his argument will be reproduced and further results presented. The worst case testing cost will then be discussed in the light of those results.

Let f be a Boolean function of a variable and let $\mathcal{M}_f$ denote the set of all minterms of f, that is

$$\mathcal{M}_f = \{(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n) = 1\} .$$

Let the weight of an input vector, $w(x_1, \ldots, x_n)$, be the sum of the components of the vector (i.e., the number of components that are equal to one). In a decision tree representation of f, a leaf labelled 1 at depth k represents 2^{n-k} minterms (since n - k variables are unspecified); in particular, if the tree has height h, then 2^{n-h} divides $|\mathcal{M}_f|$.

Consider the generating function

$$g_f(z) = \sum_{i=0}^{n} a_i z^i , \qquad (4.3\text{-}1)$$

where a_i is the number of minterms of weight i. Then

$$g_f(1) = \sum_{i=0}^{n} a_i = |\mathcal{M}_f| .$$

A leaf at depth k contributes a summand $z^{\ell}(1 + z)^{n-k}$ to $g_f(z)$, where ℓ is the weight of the k-subvector. (Since the remaining n - k components are unspecified, each can take the value 0 or 1, corresponding to the 1 and the z in the term $(1 + z)^{n-k}$.) Thus a tree of height h has each leaf labelled with a 1 contributing a multiple of $(1 + z)^{n-h}$.

Lemma 4.1. If the worst case number of tests for a function f is h, then $(1 + z)^{n-h}$ divides $g_f(z)$. □

As a special case, setting z = 1 gives the result stated above, that is, 2^{n-h} divides $|\mathcal{W}_f|$. Now, letting z = -1 yields $g_f(-1) = 0$ for h < n, so that the sum of the odd power terms is equal to that of the even power terms.

Lemma 4.2. Any Boolean function which is not exhaustive has an equal number of odd and even weight minterms. □

But now, the number of Boolean functions of n variables with an equal number of odd and even weight minterms is

$$\mathcal{N}(n) = \sum_{j=0}^{2^{n-1}} \binom{2^{n-1}}{j}^2 = \binom{2^n}{2^{n-1}}, \qquad (4.3\text{-}2)$$

which, for large n, becomes approximately $2^{2^n}/\sqrt{\pi \cdot 2^{n-1}}$ (Knuth 73, p. 71). Since the number of nonexhaustive Boolean functions must be less than $\mathcal{N}(n)$ by Lemma 4.2, and since there are 2^{2^n} Boolean functions of n variables, the fraction of all Boolean fractions that are nonexhaustive must be less than $\mathcal{N}(n)/2^{2^n}$. But, for large n:

$$\lim_{n\to\infty} \mathcal{N}(n)/2^{2^n} = \lim_{n\to\infty} \left(2^{2^n}/\sqrt{\pi \cdot 2^{n-1}}\right)/2^{2^n} = \lim_{n\to\infty} 1/\sqrt{\pi \cdot 2^{n-1}} = 0 \qquad (4.3\text{-}3)$$

which proves the following.

Theorem 4.1. Almost all Boolean functions are exhaustive. □

The above theorem is the announced result from (Rivest 74).

It is natural to suspect that there exist large groups of exhaustive functions. It will now be shown that symmetric and threshold functions are exhaustive.

A Boolean function of n variables, $f(x_1, \ldots, x_n)$, is said to be symmetric if and only if, for each permutation, σ, over n letters:

$$f(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) = f(x_1, \ldots, x_n) .$$

Equivalently, a function is symmetric if and only if there exists a set of k numbers ($k \leq n$), $\{a_1, \ldots, a_k\}$ where $0 \leq a_1 < \ldots < a_k \leq n$, such that the function is equal to one exactly when a_i of its variables are equal to one, for some i, $1 \leq i \leq k$. Such a function has a single tree structure, since testing one variable rather than another does not change the resulting subfunctions; in particular, after testing $n - a_1$ variables and finding them all equal to zero, the remaining a_1 variables must all be tested, since the function will be equal to one if all are equal to one. This proves the following result.

Theorem 4.2. All symmetric Boolean functions are exhaustive. □

Let P be the defining property of a class of functions such that, if f possesses P, then both $f_{(x_i=0)}$ and $f_{(x_i=1)}$ possess P, for any choice of x_i; in other words, P is preserved by Shannon's decomposition. All functions in the class are then exhaustive if and only if they have no redundant variables and, in any Shannon's decomposition, at least one of the two subfunctions has no redundant variables.

A class of functions of considerable interest is that of unate functions, that is, functions for which a Boolean formula can be written which uses no variable in both complemented and uncomplemented form. Since decision trees are invariant under complementation of variables, it can be assumed without loss of generality that all variables are used uncomplemented; this defines the class of positive unate functions, which can be shown to be monotone increasing (Harrison 65). Both properties are clearly preserved by Shannon's decomposition.

Let then $f(x_1, \ldots, x_n)$ be an intrinsic positive unate function; from the above, f is exhaustive if and only if, for each choice of x_i, either $f_{(x_i=0)}$ or $f_{(x_i=1)}$ is intrinsic, that is, there cannot be found x_j, x_k $(j, k \neq i)$ such that $f_{(x_i=0)}$ does not depend on x_j and $f_{(x_i=1)}$ does not depend on x_k. Without loss of generality, let i = 1, j = 2, and k = 3, and let $\underline{x}$ stand for $(x_4, \ldots, x_n)$. Then f is not exhaustive if and only if

$$f(0, 0, x_3, \underline{x}) = f(0, 1, x_3, \underline{x})$$

and

$$f(1, x_2, 0, \underline{x}) = f(1, x_2, 1, \underline{x}) \ . \qquad (4.3\text{-}4)$$

Since f is monotone increasing, it must be the case that

$$f(1, x_2, 1, \underline{x}) \geq f(0, 0, x_3, \underline{x}) \ ,$$

so that, by topological sorting, the following relations are obtained:

$$f(1, 1, 1, \underline{x}) = f(1, 1, 0, \underline{x}) \geq f(1, 0, 1, \underline{x}) = f(1, 0, 0, \underline{x})$$
$$\geq f(0, 1, 1, \underline{x}) = f(0, 0, 1, \underline{x})$$
$$\geq f(0, 1, 0, \underline{x}) = f(0, 0, 0, \underline{x}) \ . \qquad (4.3\text{-}5)$$

Let the four pairs of function points in (4.3-5) be denoted a, b, c, and d in this order. For any given choice of $\underline{x}$, these four pairs can be mapped to the same value or to two distinct values (0 and 1), with the following partitions:

(1) (abcd) mapped to the same value; then x_1, x_2, and x_3 are redundant for that choice of $\underline{x}$;

(2) (abc) mapped to 1 and (d) to 0; then x_2 is redundant for that choice of $\underline{x}$;

(3) (ab) mapped to 1 and (cd) to 0; then x_2 and x_3 are redundant for that choice of $\underline{x}$;

(4) (a) mapped to 1 and (bcd) to 0; then x_3 is redundant for that choice of $\underline{x}$.

No other choice is possible due to the monotone property. This shows that all unate functions of three or less variables are exhaustive, since then there is no choice for $\underline{x}$ and one of (1)-(4) must hold, contradicting the assumption of intrinsicalness. At the same time, however, it shows how to construct a nonexhaustive unate function of four variables: it is enough to find $\underline{x}' > \underline{x}''$ such that $f(x_1, x_2, x_3, \underline{x}')$ is partitioned according to (2) and $f(x_1, x_2, x_3, \underline{x}'')$ according to (4),

since then x_2 is redundant in one case and x_3 in the other, but both are necessary overall. One such function is given by the formula

$$f(x_1, x_2, x_3, x_4) = x_1x_2 + x_1x_4 + x_3x_4$$

and it is easily verified that it has tree representations of height 3.

Thus, not all unate functions are exhaustive, but what of a more restricted class? A special case of unate functions is that of threshold (or linearly separable) functions, where a Boolean function of n variables, $f(x_1, \ldots, x_n)$, is a threshold function if and only if there exist a set of weights, $(w_1, \ldots, w_n)$, and a threshold, T, such that $f(x_1, \ldots, x_n)$ is one exactly when

$$\sum_{i=0}^{n} w_i x_i \geq T \ .$$

Since unate functions can be taken to be positive, weights and threshold are assumed positive without loss of generality. Let now $\underline{w}$ stand for $(w_4, \ldots, w_n)$; substituting weights and threshold into the four pairs of (4.3-5) yields:

$$a = w_1 + w_2 + \underline{w}\cdot\underline{x}^t;\ w_1 + w_2 + w_3 + \underline{w}\cdot\underline{x}^t);$$

$$b = (w_1 + \underline{w}\cdot\underline{x}^t;\ w_1 + w_3 + \underline{w}\cdot\underline{x}^t);$$

$$c = (w_3 + \underline{w}\cdot\underline{x}^t;\ w_2 + w_3 + \underline{w}\cdot\underline{x}^t);$$

$$d = (\underline{w}\cdot\underline{x}^t;\ w_2 + \underline{w}\cdot\underline{x}^t) \ ,$$

where $\underline{x}^t$ stands for the transpose of $\underline{x}$. As seen above, a function will not be exhaustive only if $\underline{x}' > \underline{x}''$ can be found such that $f(x_1, x_2, x_3, \underline{x}')$ is partitioned as (abc)(d) and $f(x_1, x_2, x_3, \underline{x}'')$ as (a)(bcd). Using only boundary values around the threshold, this implies, for the first partition:

$$w_1 + w_2 + \underline{w} \cdot \underline{x}'^t \geq T,\ w_1 + w_3 + \underline{w} \cdot \underline{x}'^t < T,\ w_2 + w_3 + \underline{w} \cdot \underline{x}'^t < T\ ,$$

and for the second:

$$w_3 + \underline{w} \cdot \underline{x}''^t \geq T,\ w_1 + \underline{w} \cdot \underline{x}''^t \geq T,\ w_2 + \underline{w} \cdot \underline{x}''^t < T\ .$$

The first three inequalities yield $w_2 > w_3$ and the second three yield $w_2 < w_3$, a contradiction. Hence $\underline{x}'$ and $\underline{x}''$ cannot be found, which proves the following.

Theorem 4.3. All linearly separable functions are exhaustive. □

The above results show that the worst case testing cost does not discriminate between most Boolean functions. Thus, although it can be efficiently minimized [the dynamic programming algorithm of (Bayes 73) can be applied], the worst case testing cost must be rejected as a measure of the complexity of decision trees.

4.3.4. The expected testing cost

By successive eliminations, then, the expected testing cost, E, is selected as the most representative measure of the complexity of decision trees. This section establishes a few simple results about this measure and examines the complexity of its optimization.

Given an intrinsic Boolean function of n variables, $f(x_1, \ldots, x_n)$, where testing variable x_i incurs cost c_i, and given a probability distribution, p, on the input combinations, the expected testing cost of any tree representation, T, of f is obviously bounded by:

$$\min \{ c_i \mid i = 1, \ldots, n\} < E(T) \leq \sum_{i=1}^{n} c_i \ . \qquad (4.3\text{-}6)$$

In particular, if p is uniform and all the costs unity, then:

$$n/2^n + \sum_{i=1}^{n} i/2^i = 2 - 2^{-(n-1)} \leq E(T) \leq n \ .$$

In fact, Boolean functions with unity costs and uniform probability distributions require an expected number of tests that converges to n; this can be shown as follows. Let F(n) be the number of Boolean functions of n variables and let I(n) be the number of those that are intrinsic; thus:

$$F(n) = 2^{2^n}$$

and

$$I(n) = \sum_{i=0}^{n} (-1)^{n-i} \cdot \binom{n}{i} \cdot F(i) \ .$$

As remarked in Section 2.2.1, almost all Boolean functions are intrinsic, that is:

$$\lim_{n\to\infty} I(n)/F(n) = 1 ,$$

with rapid convergence. Now, the expected value of E for a function of n variables, E(n), must be at least as large as E(n - 1) for nonintrinsic functions and equal to 1 + E(n - 1) otherwise; hence the recurrence:

$$E(n) \geq [F(n) - I(n) \cdot E(n-1) + I(n) \cdot (1 + E(n-1))]/F(n)$$
$$= E(n-1) + I(n)/F(n) . \qquad (4.3\text{-}7)$$

Since I(n)/F(n) rapidly converges to 1 (for n = 4, it is already within two percent), the expected value of E is essentially equal to n for large values of n.

The above result, however, does not indicate that minimizing the expected testing cost is useless, since the presence of nonuniform costs and probabilities results in the wide range of values described by (4.3-6). Moreover, as seen in Section 3.3, this minimization can be accomplished very efficiently by dynamic programming--in time $\Theta(s^{\log_2 3} \cdot \log s)$ for an input of size $s = \Theta(2^n)$. This result has often been misinterpreted, most recently in (Standish 80, p. 176), by taking n as the input size and declaring the algorithm to be exponential time since it requires $\Theta(n \cdot 3^{n-1})$ operations. An exception is the case where the probability distribution is only incompletely specified and the function itself given by its minterms or by a simplified formula (as in decision tables where probabilities are specified only for--not necessarily simple--rules), resulting in an input, the size of which may be polynomial in n. In that case, however, there is no properly optimal

tree (because the expected testing cost cannot be computed exactly, but can only be bounded) unless the combinations grouped in a single probability assignment are assumed equally likely, in which case it is conjectured that the optimization problem is NP-hard (and thus NP-equivalent, since it is clearly NP-easy).

4.4. The General Case

In the general case, the function given may be partial only. Then, even in the case of Boolean functions with unity costs and uniform distributions, no two measures are compatible (except, possibly, the external path length and the number of nodes: the implication η_{min} ==> α_{min} could neither be proved nor disproved, although it is clearly false with arbitrary costs or probability distributions). The reasons enounced in the previous section for choosing testing costs rather than storage costs as measures of the complexity of decision trees remain valid, as does the selection of the expected testing cost over the other measures of testing cost.

The dynamic programming algorithm for the minimization of the expected testing cost is applicable to the general case, so that the complexity of optimizing E is polynomial when the probability distribution is completely specified, and probably NP-equivalent when inputs are assumed to be equally likely and the function is partial. A further study of the expected testing cost will be presented in the next chapter.

CHAPTER 5

ACTIVITY OF A VARIABLE

5.1. Introduction

In the previous chapter, the expected testing cost was chosen as the measure of complexity for decision trees and diagrams. This does not imply that the minimum expected testing cost for a function should be chosen as a measure of that function's complexity; for instance, such a measure would not be implementation-independent. It does mean, however, that the relationship between a function and the expected testing costs of its decision tree representations must be investigated. In particular, as pointed out in Chapter 1, a characterization of the influence of individual variables on the measure is desirable.

The following sections develop such a characterization, the activity of a variable, and show its relation to decision trees. The concept is then extended to hierarchies of recursive relations and its application to conventional problems (such as the heuristic construction of suboptimal trees) and to the development of a measure of function complexity is investigated.

5.2. Definition and Results

The contribution of any variable to the expected testing cost of a tree varies between zero and that variable's testing cost. Clearly, a redundant variable, although it may be used in a tree, is not a priori expected to make any contribution since it need not be tested.

Conversely, an indispensable variable, that is, one which, regardless of the values of the other variables, must be tested in order to determine the value of the function, is expected to contribute its entire testing cost. Hence, an a priori measure of the influence of a variable on the expected testing cost of any tree representation must vary between zero--for a redundant variable--and that variable's testing cost--for an indispensable variable.

Moreover, such a measure should be compatible with the decomposition process characteristic of decision trees. That is, the measure on the whole function should be related to that of the subfunctions in much the same way as the expected testing cost is. This leads to the following definition.

Definition 5.1. Given a (partial) function of n variables, $f(x_1, \ldots, x_n)$, where each variable, x_i, takes on m_i values and has an associated testing cost of c_i, and where the probability of an input vector, $(x_1, \ldots, x_n)$, is denoted $p(x_1, \ldots, x_n)$, the activity, $a_f(x_i)$, of variable x_i with respect to the function f is defined by the two relations:

(a) $0 \leq a_f(x_i) \leq c_i$;

moreover, $a_f(x_i) = 0$ if and only if x_i is redundant, and $a_f(x_i) = c_i$ if and only if x_i is indispensable;

(b) for any x_j, $j \neq i$,

$$a_f(x_i) = \sum_{k=0}^{m_j-1} p(x_j = k) \cdot a_{f_{(x_j=k)}}(x_i) \ . \qquad \square$$

The second relation requires that the activity of a variable with respect to a function be equal to the expected value of the activities of that variable with respect to the subfunctions resulting from a decomposition.

Exactly one function can satisfy Definition 5.1. This can be seen by examining the case of functions of two variables. Let $f(x_1, x_2)$ be such a function and let it be decomposed around x_2. The resulting subfunctions, $f_{(x_2=k)}$ for $k = 0, \ldots, m_2 - 1$, are functions of a single variable, x_1. Consequently, that variable is either redundant or indispensable and thus, by Definition 5.1(a), its activity is either null or equal to its cost. Hence, by Definition 5.1(b), the activity of x_1 with respect to f is

$$a_f(x_1) = \sum_k p(x_2 = k) \cdot c_1 = c_1 \cdot \sum_k p(x_2 = k)$$

where the sum is taken over all k such that $f(x_1, k)$ depends on x_1. A straightforward induction argument then completes the proof of the following result.

Theorem 5.1. Exactly one function satisfies the definition of activity:

$$a_f(x_i) = c_i \cdot \sum p(x_1 = k_1, \ldots, x_{i-1} = k_{i-1}, x_{i+1} = k_{i+1}, \ldots, x_n = k_n) = c_i \cdot p_f^+(x_i) \ ,$$

where the sum is taken over all values, $k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n$, such that, for some pair of values (v_1, v_2) of x_i, $f(k_1, \ldots, k_{i-1}, v_1, k_{i+1}, \ldots, k_n)$ and $f(k_1, \ldots, k_{i-1}, v_2, k_{i+1}, \ldots, k_n)$ are specified and different. □

The quantity, $p_f^+(x_i)$, will be called the probability that f is sensitized to x_i; it is the a priori probability that x_i will be needed in testing for the values of the function. Conversely, the a priori probability that x_i will be useless, $1 - p_f^+(x_i)$, will be denoted $p_f^-(x_i)$.

As defined above, the activity is a generalization of a concept developed from Boolean calculus in (Bozoyan 75). When the function is a completely specified Boolean function, the activity can be written as

$$a_f(x_i) = c_i \cdot p\left(\frac{\partial f}{\partial x_i} = 1\right),$$

where $\partial f/\partial x_i$ is the Boolean difference of f with respect to x_i. [For details on the Boolean difference, see (Thayse 73).] *If it is further* assumed that the probability distribution is uniform, then the activity can be expressed as

$$a_f(x_i) = c_i \cdot \zeta_f(x_i)/2^n ,$$

where $\zeta_f(x_i)$ denotes the Chow parameter of x_i with respect to f (Winder 71).

Since the activity of a variable is an a priori measure of its contribution to the expected testing cost, the difference between the

actual contribution and the activity is a measure of the loss incurred by testing the variable. In particular, if the variable is tested at the root, its actual contribution is equal to its testing cost, so that the loss incurred becomes

$$\ell_f(x_i) = c_i - a_f(x_i) = c_i - c_i \cdot p_f^+(x_i) = c_i \cdot p_f^-(x_i) \ .$$

Example 5.1. Let f be the partial function of Example 2.10, given by

f:	(0, 0, 0) → a	(1, 0, 0) → a	(2, 0, 1) → b
	(0, 0, 1) → a	(1, 0, 1) → b	(2, 1, 0) → a
	(0, 1, 1) → c	(1, 1, 0) → c	(2, 1, 1) → b
p:	(0, 0, 0) → 0.00	(1, 0, 0) → 0.10	(2, 0, 0) → 0.00
	(0, 0, 1) → 0.05	(1, 0, 1) → 0.05	(2, 0, 1) → 0.20
	(0, 1, 0) → 0.00	(1, 1, 0) → 0.10	(2, 1, 0) → 0.10
	(0, 1, 1) → 0.10	(1, 1, 1) → 0.15	(2, 1, 1) → 0.15

and the testing costs, $c_1 = 5$, $c_2 = 3$, $c_3 = 2$. The probability of f being sensitized is computed for each variable as follows:

$$\begin{aligned} p_f^+(x_1) &= p(x_2 = 0, x_3 = 1) + p(x_2 = 1, x_3 = 0) + p(x_2 = 1, x_3 = 1) \\ &= (0.05 + 0.05 + 0.20) + (0.00 + 0.10 + 0.10) \\ &\quad + (0.10 + 0.15 + 0.15) = 0.90 \ ; \end{aligned}$$

$$\begin{aligned} p_f^+(x_2) &= p(x_1 = 0, x_3 = 1) + p(x_1 = 1, x_3 = 0) \\ &= (0.05 + 0.10) + (0.10 + 0.10) = 0.35 \ ; \end{aligned}$$

$$p_f^+(x_3) = p(x_1 = 1, x_2 = 0) + p(x_1 = 2, x_2 = 1)$$

$$= (0.10 + 0.05) + (0.10 + 0.15) = 0.40 .$$

Then the other quantities defined above are:

$$p_f^-(x_1) = 0.10, \; p_f^-(x_2) = 0.65, \; p_f^-(x_3) = 0.60 ;$$

$$a_f(x_1) = 5 \cdot 0.90 = 4.50, \; a_f(x_2) = 3 \cdot 0.35 = 1.05, \; a_f(x_3) = 2 \cdot 0.40$$

$$= 0.80 ;$$

$$\ell_f(x_1) = 0.50, \; \ell_f(x_2) = 1.95, \; \ell_f(x_3) = 1.20 . \qquad \square$$

The concepts of activity and loss are closely related to the expected testing cost. As observed above, each variable makes a minimal contribution equal to its activity; then a loss is added each time the variable is tested. This relationship is formalized below.

Theorem 5.2. The expected testing cost, E(T), of a decision tree, T, for the function f can be expressed as

$$E(T) = \sum_{i=1}^{n} a_f(x_i) = \sum_k p(f_k) \cdot \ell_{f_k}(x_k) ,$$

where the second sum is taken over all internal nodes and x_k and f_k refer to the variable and the subfunction associated with the k-th node.

Proof: The proof is by induction on n, the number of variables. For n = 1, the basis is easily verified: the variable space is just

an m-tuple and there are only two possible tree structures. Assume that the theorem holds for all functions of up to and including n - 1 variables and choose x_i to be the root of T. This determines m_i subfunctions, each of n - 1 variables, so that the inductive hypothesis applies and, for each subfunction $f_{(x_i=j)}$, j = 0, ..., m_i - 1, with corresponding tree T_j, the expected testing cost can be written as

$$E(T_j) = \sum_{\substack{k=1 \\ k\neq i}}^{n} a_{f_{(x_i=j)}}(x_k) + \sum_s p\left(f_{(x_i=j)_s}\right) \cdot \ell_{f_{(x_i=j)_s}}(x_s) ,$$

where the second sum is taken over all internal nodes, s, of T_j. But,

$$E(T) = c_i + \sum_{j=0}^{m_i-1} p\left(f_{(x_i=j)}\right) \cdot E(T_j)$$

and, upon substitution, this becomes

$$E(T) = c_i - \ell_f(x_i) + \sum_{j=0}^{m_i-1} \left[p\left(f_{(x_i=j)}\right) \cdot \sum_{\substack{s=1 \\ s\neq i}}^{n} a_{f_{(x_i=j)}}(x_s) \right]$$

$$+ \sum_k p(f_k) \cdot \ell_{f_k}(x_k) ,$$

where the last sum is taken over all internal nodes, k, of T. However, the following relations obviously hold:

$$\sum_{j=1}^{n} a_f(x_j) = a_f(x_i) + \sum_{j=0}^{m_i-1} \left[p\left(f_{(x_i=j)}\right) \cdot \sum_{\substack{s=1 \\ s\neq i}}^{n} a_{f_{(x_i=j)}}(x_s) \right] ,$$

and

$$c_i - \ell_f(x_i) = a_f(x_i) .$$

Substitution of those two equalities in the expression for E(T) yields the conclusion. □

A simplified form of this theorem was proved in (Breitbart 75b), using Chow parameters, for completely specified monotone Boolean functions of uniformly distributed variables with unity costs.

Corollary 5.1. The expected testing cost of any decision tree, T, for the function f having x_i as root test is bounded by

$$\sum_{j=1}^{n} c_j \leq E(T) \leq \ell_f(x_i) + \sum_{j=1}^{n} a_f(x_j) . \qquad \square$$

This corollary, in simplified form, was proved in (Reinwald 66) and is implicit in (Breitbart 75b). Both references use it as the basis for a branch-and-bound search algorithm to find a tree with minimal expected testing cost.

Those results stress the importance of the sum of the activities of the variables as an implementation-independent measure of the cost incurred in determining the values of a function. This motivates the following definition.

Definition 5.2. The intrinsic cost, I(f), of the function f is the quantity:

$$I(f) = \sum_{i=1}^{n} a_f(x_i) \ . \qquad \square$$

Example 5.2. Consider again the function of Example 5.1. Its intrinsic cost is

$$I(f) = 4.50 + 1.05 + 0.80 = 6.35 \ .$$

From Corollary 5.1, the lower bound on the expected testing cost of any tree can be computed for each choice of root:

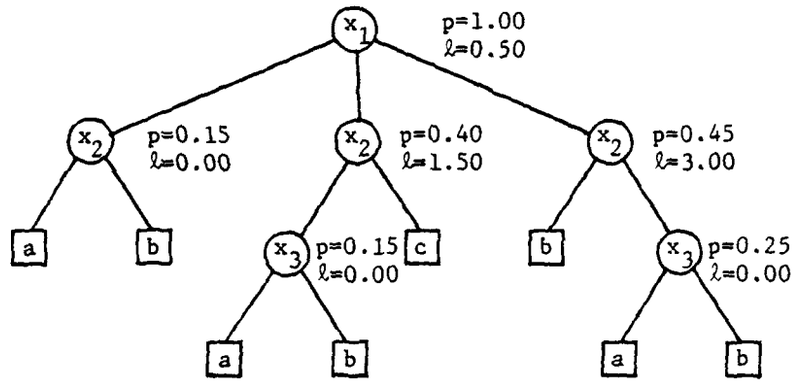
$$lb(x_1) = 6.35 + 0.50 = 6.85 \ ;$$

$$lb(x_2) = 6.35 + 1.95 = 8.30 \ ;$$

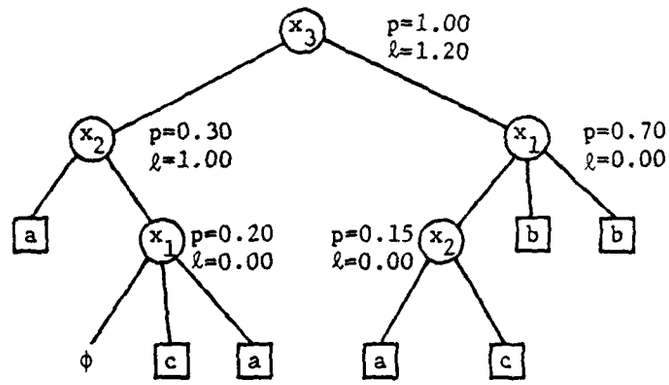
$$lb(x_3) = 6.35 + 1.20 = 7.55 \ .$$

Theorem 5.2 is illustrated by considering the two trees of Figure 2.5 (page 31), which are reproduced in Figure 5.1 with the loss and probability of each internal node. The expected testing cost of tree (a) is then

$$E_{(a)} = 6.35 + 1.00 \cdot 0.50 + 0.15 \cdot 0.00 + 0.40 \cdot 1.50$$
$$+ 0.45 \cdot 3.00 + 0.15 \cdot 0.00 + 0.25 \cdot 0.00 = 6.35 + 0.50$$
$$+ 0.60 + 1.35 = 8.80$$



(a)



(b)

Figure 5.1. The two trees for Example 5.2 with the probability (p) and loss (ℓ) at each internal node.

and that of tree (b) is

$$E_{(b)} = 6.35 + 1.00 \cdot 1.20 + 0.30 \cdot 1.00 + 0.70 \cdot 0.00$$
$$+ 0.20 \cdot 0.00 + 0.15 \cdot 0.00 = 6.35 + 1.20 + 0.30 = 7.85 ,$$

which are the values found in Example 2.10. □

5.3. Extension to Recursive Relations

In this section, the concepts of decision tree and activity are extended to relations, recursive relations, and hierarchies of relations. This allows the modelling of systems with simple feedback loops, and systems composed of several subsystems. In the case of decision tables, this makes it possible to consider ambiguous tables, recursive tables, and tables incorporating calls to subtables in place of actions (each of which is beyond the reach of published analyses).

The extension to relations on finite sets is of particular interest in the case of interdependent functions which must be represented by a single tree [as in (Cerny 79)] and in the case of ambiguous decision tables tables using the interpretation of (King 73).

A relation, R, from the set of input vectors to the set of output values, might specify no more than one output for each input combination, in which case it is a (partial) function; it may, however, specify more than one output, in which case it can be arbitrarily decided to specify any particular output or subset of outputs. For consistency of notation, it will be assumed that an unspecified entry is in fact related to the whole output set, so that any subset of output values may be specified

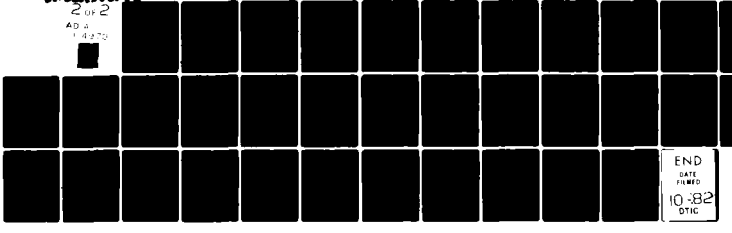
for such an input combination. Such a relation can obviously be represented by decision trees and diagrams. Testing costs are defined as for functions, and so are probability distributions. (Thus, the distribution depends only on the input values and is unaffected by the choice of one or another output subset.) The concepts of activity and loss are then generalized in the obvious way by modifying the definition of $p_R^+(x_i)$, modification which results from the fact that two output subsets specified by the relation are considered different if their intersection is empty. It is readily verified that, under these assumptions, all results previously stated for partial functions remain valid for relations.

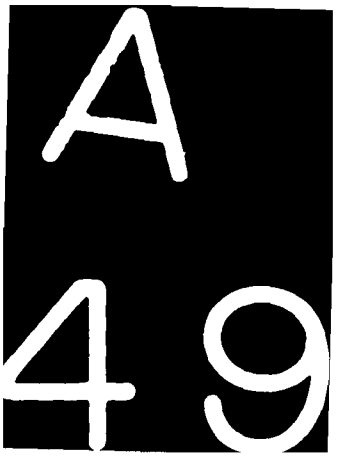
Example 5.3. Let R be the relation from the input set $\{0, 1, 2\}^2 \times \{0, 1\}$ to the output set $\Omega = \{a, b, c, d\}$, where all three variables have unity costs and the relation and the probability distribution are given by:

R:	(0, 0, 1) → {a}	(1, 0, 0) → {b, d}	(2, 0, 0) → {d}
	(0, 1, 0) → {a, b, c}	(1, 0, 1) → {b}	(2, 0, 1) → {b}
	(0, 1, 1) → {a}	(1, 1, 0) → {b, c, d}	(2, 1, 0) → {c}
		(1, 1, 1) → {a}	(2, 1, 1) → {c}
p:	(0, 0, 0) → 0.05	(1, 0, 0) → 0.10	(2, 0, 0) → 0.00
	(0, 0, 1) → 0.10	(1, 0, 1) → 0.10	(2, 0, 1) → 0.05
	(0, 1, 0) → 0.10	(1, 1, 0) → 0.10	(2, 1, 0) → 0.05
	(0, 1, 1) → 0.20	(1, 1, 1) → 0.05	(2, 1, 1) → 0.10

AD-A114 970 TENNESSEE UNIV KNOXVILLE DEPT OF ELECTRICAL ENGINEERING F/G 12/1
THE REPRESENTATION OF DISCRETE FUNCTIONS BY DECISION TREES.(U)
FEB 82 R C GONZALEZ, M G THOMASON, B M MORET N00014-78-C-0311
UNCLASSIFIED TR-EE/CS-82-20 NL

2 OF 2
AD A
114 970




A
49

Since all variables have unity costs, $a_R(x_i) = p_R^+(x_i)$, so that:

$$a_R(x_1) = p(x_2 = 0,\ x_3 = 1) + p(x_2 = 1,\ x_3 = 1)$$

$$= (0.10 + 0.10 + 0.05) + (0.20 + 0.05 + 0.10) = 0.60 \ ;$$

similarly, $a_R(x_2) = 0.35$ and $a_R(x_3) = 0.20$. Hence, the intrinsic cost is

$$I(R) = 0.60 + 0.35 + 0.20 = 1.15 \ .$$

Choosing x_3 as the root test for a decision tree results in a lower bound on the expected testing cost of 1.15 + (1-0.6) = 1.55. A possible decision tree, T, rooted in x_3 is shown in Figure 5.2 together with the losses of its nodes. Its expected testing cost is

$$E(T) = 1.15 + 0.35 \cdot 4/7 = 1.75 \ ,$$

and it is in fact one of the optimal trees for R. □

As a further extension to the foregoing concepts, the case of recursive functions or relations is now considered. A recursive relation is a relation that, for certain input vectors, does not specify output values, but calls for a new evaluation of itself. In the following discussion, it is assumed that an unspecified entry can only be replaced by a subset of output values (not by a recursive call). This allows the computation of the probability, e, that an evaluation will be made without recursive calls; if e is one, then the relation is not recursive, while if e is zero, then the relation will never yield a value, but will keep issuing recursive calls ad infinitum.

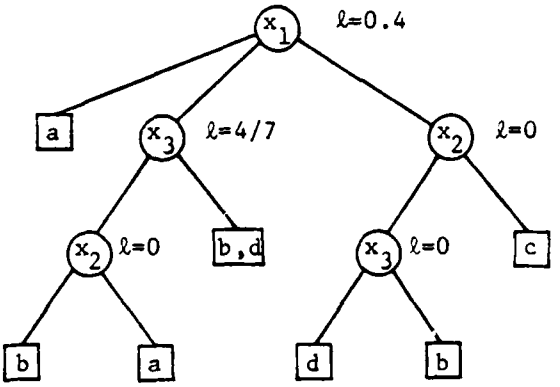


Figure 5.2. The decision tree of Example 5.3 with the losses of its nodes.

A recursive relation can be represented by an infinite decision tree, where each recursive call leads to the root of a new component tree; it will be assumed that all component trees have the same structure. Under this assumption, it is also possible to represent a recursive relation by a diagram with cycles, each cycle leading back to the root of the diagram.

A first question about such representations concerns an upper bound on their testing cost. Such a bound was set by Corollary 5.1 for nonrecursive relations as the sum of the testing costs of the variables, but can evidently be passed by recursive relations. The following proposition provides the answer.

Proposition 5.1. Let R be a recursive relation on n variables, $x_1, \ldots, x_n$, with costs $c_1, \ldots, c_n$, and let e be the probability that no recursive call will be needed; then the expected testing cost of any representation of R is no larger than

$$(1/e) \cdot \sum_{i=1}^{n} c_i \ .$$

Proof: The probability of a recursive call occurring in any evaluation is (1 - e); an evaluation results in, at worst, the test of all variables, for a cost of

$$\sum_{i} c_i \ ;$$

thus the total cost is no larger than

$$\sum_{k=0}^{\infty} (1 - e)^k \cdot \sum_{i=1}^{n} c_i = (1/e) \cdot \sum_{i=1}^{n} c_i \ . \qquad \square$$

The actual cost of a tree representation can be computed by solving a simple first degree equation; the probability that the relation will take on a specific value can be obtained from a similar equation, with the provision that entries for which several values are specified are set to the specific value under consideration wherever possible.

The activity is generalized to recursive relations by re[illegible]g its computation for relations on one variable as follows. If the rela[illegible]n is not recursive, the activity is that defined previously, otherwise, it is equal to the product of the variable's testing cost times the probability that the relation will take on more than one value. This quantity will be called the tree activity; the corresponding loss, the tree loss, is equal to the testing cost of the variable minus its tree activity. The same quantities multiplied by (1/e) will be referred to as the diagram activity and diagram loss.

Theorem 5.3. Let R be a recursive relation and let a decision procedure for F be represented by a diagram, D, and by an infinite tree, T. The expected testing cost of the procedure, E(D) = E(T), is equal to (a) the sum of the diagram activities and that, taken over all internal nodes of the diagram, of the diagram losses, or (b) the sum, taken over the infinite tree, of the tree activities and of the tree losses.

Proof: The proof relies on that of Theorem 5.2 and on simple considerations on the series 1, $(1 - e)$, $(1 - e)^2$, ..., and its sum, $1/e$. If, in the diagram, D, all recursive calls are replaced by leaves, the cost of the resulting tree is the sum of the tree activities and that, taken over all internal nodes of the tree, of the tree losses. Introducing recursion (either as a diagram or as an infinite tree) results in a series of invocations, the probabilities of which are described by the series $(1 - e)^k$. □

Corollary 5.1 is similarly extended.

Finally, all of the above results can easily be extended to hierarchies of (recursive) relations by analyzing a hierarchy component by component and putting the results together using the probability of each component. It is noted, however, that the results are not directly extendable to indirect recursion (that is, through a chain of relations) unless all recursive calls are to the same relation at the top of the hierarchy. This latter case is now demonstrated along with the other extensions in a practical example.

Example 5.4. Consider a situation in which a monitoring program must periodically evaluate several system variables. If the sampled values point to a satisfactory status, the program waits for a specific period of time and examines the variables again. Otherwise, either a malfunction is detected and the program takes some action and stops, or further analysis is required and additional variables are examined to determine whether the program should resume its normal cycle or take some action and stop. The first part of the examination (the normal cycle) is described by the relation R1, which includes calls both to itself and to the second relation, R2 (the exception cycle), which includes calls to R1 (thereby producing indirect recursion).

In this simple example, R1 is a relation between $\{0, 1, 2\} \times \{0, 1\}^2$ and the set of actions, $\Omega = \{a, b\}$, and R2 is a relation between $\{0, 1\}^3$ and Ω, as specified below, together with the probability distributions, p_1 and p_2. The variables will be denoted x_1, x_2, and x_3 for R1, and y_1, y_2, and y_3 for R2; their testing costs are listed below.

R1:	$(0, 0, 0) \rightarrow \{R1\}$	$(1, 0, 0) \rightarrow \{R1\}$	$(2, 0, 0) \rightarrow \{b\}$
	$(0, 0, 1) \rightarrow \{R1\}$	$(1, 0, 1) \rightarrow \{a\}$	$(2, 0, 1) \rightarrow \{R2\}$
	$(0, 1, 0) \rightarrow \{R1\}$	$(1, 1, 0) \rightarrow \{R1\}$	$(2, 1, 1) \rightarrow \{R2\}$
	$(0, 1, 1) \rightarrow \{R1\}$	$(1, 1, 1) \rightarrow \{R2\}$	

p_1:	$(0, 0, 0) \rightarrow 0.70$	$(1, 0, 0) \rightarrow 0.05$	$(2, 0, 0) \rightarrow 0.01$
	$(0, 0, 1) \rightarrow 0.05$	$(1, 0, 1) \rightarrow 0.01$	$(2, 0, 1) \rightarrow 0.01$
	$(0, 1, 0) \rightarrow 0.05$	$(1, 1, 0) \rightarrow 0.04$	$(2, 1, 0) \rightarrow 0.01$
	$(0, 1, 1) \rightarrow 0.05$	$(1, 1, 1) \rightarrow 0.01$	$(2, 1, 1) \rightarrow 0.01$

R2: (0, 0, 0) → {R1} (1, 0, 0) → {R1}

(0, 0, 1) → {b} (1, 0, 1) → {a}

(0, 1, 0) → {a} (1, 1, 0) → {a}

(0, 1, 1) → {b}

p_2: (0, 0, 0) → 0.25 (1, 0, 0) → 0.25

(0, 0, 1) → 0.10 (1, 0, 1) → 0.10

(0, 1, 0) → 0.10 (1, 1, 0) → 0.10

(0, 1, 1) → 0.05 (1, 1, 1) → 0.05

$c(x_1) = 4.5$, $c(x_2) = 9$, $c(x_3) = 9$, $c(y_1) = 50$, $c(y_2) = 45$, $c(y_3) = 36$.

The analysis treats R1 and R2 separately and considers a structure from which all recursive calls have been removed. Once this structure has been analyzed by the methods developed above, the results are put together using p(R2), the probability that R2 is called from R1 in a given evaluation. Recursion is then taken into account by multiplying the results by (1/e), where e is the overall probability that no recursion will be needed.

p(R2) is found to be 0.01 + 0.01 + 0.01 = 0.03; similarly, p(R1), the probability that R1 will be called in a given evaluation of R2 is equal to 0.25 + 0.25 = 0.50; finally, the probability that no recursive call will be needed is the probability of directly reaching a value in R1 or through a single call to R2:

$$e = 0.01 + 0.01 + 0.01 + p(R2) \cdot (0.10 + 0.10 + 0.10 + 0.05 + 0.05 + 0.10) = 0.045 .$$

The maximum probabilities of a relation yielding a value can then be computed:

$$p(R1 = a) = \left(0.01 + 0.01 + p(R2) \cdot (0.10 + 0.10 + 0.10 + 0.05)\right) \cdot (1/e) = 0.6\overline{7} ;$$

$$p(R1 = b) = \left(0.01 + 0.01 + p(R2) \cdot (0.10 + 0.05 + 0.05)\right) \cdot (1/e) = 0.5\overline{7} ;$$

$$p(R2 = a) = 0.10 + 0.10 + 0.10 + 0.05 + p(R1) \cdot p(R1 = a) = 0.6\overline{8} ;$$

$$p(R2 = b) = 0.10 + 0.05 + 0.05 + p(R1) \cdot p(R1 = b) = 0.4\overline{8} .$$

The tree activities are:

$$a_{R1}(x_1) = c(x_1) \cdot (0.76 \cdot p(R1 \neq b) + 0.07 + 0.07 + 0.07 + 0.10) = 2.524 ;$$

$$a_{R1}(x_2) = c(x_2) \cdot (0 + 0 + 0.02 \cdot p(R2 \neq b) + 0 + 0.02 \cdot p(R2 \neq a) + 0) = 0.148 ;$$

$$a_{R1}(x_3) = c(x_3) \cdot (0 + 0.06 \cdot p(R1 \neq a) + 0.02 \cdot p(R2 \neq b) + 0 + 0.05 + 0) = 0.716 .$$

Similarly, $a_{R2}(y_1) = 10$, $a_{R2}(y_2) = 10.15$, and $a_{R2}(y_3) = 14.78$. Thus, the intrinsic cost of the relations, I, is the sum of the tree activities of R1 and of the tree activities of R2 (weighted by p(R2)) times the recursion factor, 1/e:

$$I = \left(2.524 + 0.148 + 0.716 + p(R2) \cdot (10.00 + 10.15 + 14.78)\right) \cdot (1/e) = 98.57\overline{5} .$$

The upper bound on the expected testing cost, as given by Proposition 5.1, is

$$Cmax = \left(4.5 + 9 + 9 + p(R2) \cdot (50 + 45 + 36)\right) \cdot (1/e) = 587.\overline{3} .$$

Figure 5.3 shows a possible decision diagram, D, for the relations; the lower bound for the cost of this diagram is the sum of the intrinsic cost and of the diagram loss of x_1:

$$lb(D) = 98.57\overline{5} + (4.5 - 2.524) \cdot (1/e) = 142.48\overline{6} .$$

The cost of the diagram, E(D), can be computed from Theorem 5.3(a) using the diagram losses and probabilities associated with the nodes:

$$\begin{aligned} E(D) &= 98.57\overline{5} + 1.00 \cdot 43.9\overline{1} + 0.11 \cdot 73.\overline{93} + 0.04 \cdot 148.\overline{8} \\ &\quad + 0.02 \cdot 137.\overline{7} + 0.02 \cdot 97.\overline{7} + 0.02 \cdot 200 \\ &\quad + 3 \cdot (0.01 \cdot 471.\overline{5} + 0.007 \cdot 677.\overline{7} + 0.003 \cdot 370.\overline{370}) \\ &= 197 ; \end{aligned}$$

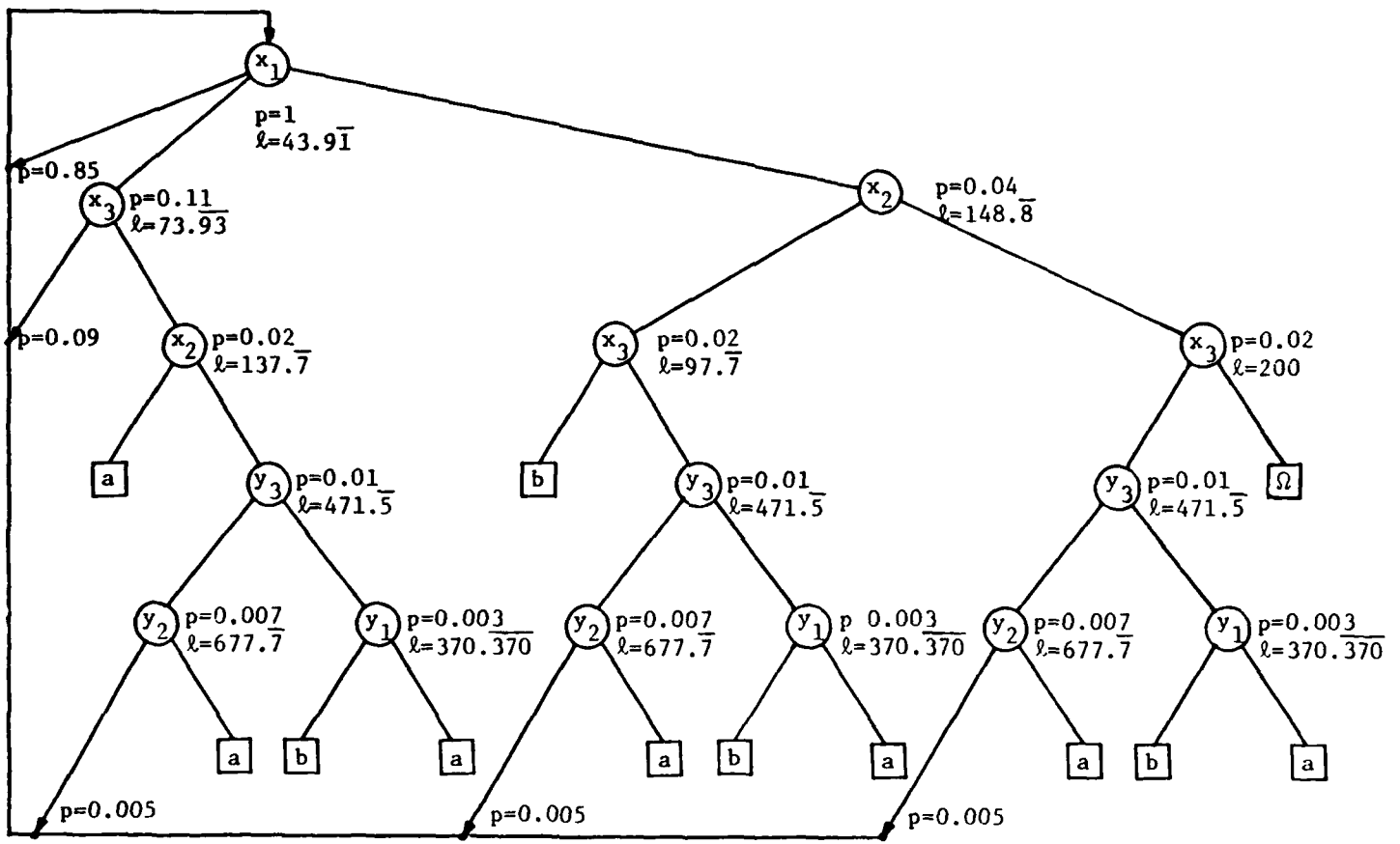


Figure 5.3. The decision diagram, D, of Example 5.4, with its diagram losses and node probabilities.

it can also be obtained by solving the linear equation expressing the cost in terms of itself:

$$\begin{aligned} E(D) = {} & 1.00 \cdot 4.5 + 0.85 \cdot E(D) + 0.11 \cdot 9 + 0.04 \cdot 9 + 0.04 \cdot 9 \\ & + 0.09 \cdot E(D) + 0.02 \cdot 9 + 0.02 \cdot 9 + 0.02 \cdot 9 \\ & + (0.01 + 0.01 + 0.01) \cdot 36 + (0.007 + 0.007 + 0.007) \cdot 45 \\ & + (0.003 + 0.003 + 0.003) \cdot 50 + (0.005 + 0.005 \\ & + 0.005) \cdot E(D) \ , \end{aligned}$$

yielding

$$(1.00 - 0.955) \cdot E(D) = 8.865 \ ,$$

so that $E(D) = 8.865/0.045 = 197$. □

5.4. Applications to Selection and Other Problems

As mentioned above, a simplified version of activity has been used as the basis for a branch-and-bound algorithm to find the tree with minimal expected testing cost (Reinwald 66, Breitbart 75b); both references pointed out that the same algorithm without backtracking was a fast, albeit suboptimal, heuristic procedure, but failed to provide an analysis of its performance.

The same algorithm is easily generalized to the case of recursive relations; it consists of the following selection rule: when developing the decision tree for the subfunction f, choose as the root the variable with the lowest loss, ℓ_f; in the case of tie, choose the variable with the lowest cost; if a tie subsists, choose any variable. The use of the

loss as a selection criterion is consistent with the requirements set forth in (Ganapathy 73). As a selection criterion, the loss has several advantages over similar criteria [such as found in (Pollack 65, Verhelst 72, Halpern 74a, Breitbart 75a)]: it is simpler to compute, more general, and optimal in several cases.

In particular, the loss criterion will always lead to the selection of an indispensable variable (since such variables have a loss of zero by definition); such a choice is easily seen to be optimal (Ganapathy 73). Also, the lower bound of Corollary 5.1 is exactly the expected testing cost when the relation is on two variables or less; it follows that the loss criterion is optimal for all (recursive) relations on two or less variables.

However, like any other method relying on local optimization, the loss criterion can lead to pessimal solutions. The following example illustrates the worst case behavior of the loss criterion for completely specified Boolean functions with unity costs.

Let f be the Boolean function given by the formula

$$f = x_1 + \bigoplus_{i=2}^{n} x_i \ ,$$

where $\oplus$ denotes summation modulo 2, and assume the following probability distribution:

each input vector satisfying $x_1 \cdot \overline{\bigoplus_{i=2}^{n} x_i} = 1$ has probability

$\gamma \cdot \varepsilon$, for $\gamma < 1$, $\gamma \simeq 1$;

each input vector satisfying $x_1 = 0$ has probability ε ;

all other input vectors have probability $\alpha = 2^{2-n} - (\gamma + 2)\cdot\varepsilon$.

Then the activity of x_1 is

$$a_f(x_1) = 2^{n-2} \cdot (\gamma + 1) \cdot \varepsilon$$

and that of any other variable, x_i, is

$$a_f(x_i) = 2^{n-1} \cdot \varepsilon ,$$

so that $\ell_f(x_i) < \ell_f(x_1)$. The two subfunctions resulting from the choice of some x_i, $i \neq 1$, as the root test are again of the form $x_1 + \oplus x_j$, so that the trees constructed by the loss criterion test variable x_1 last of all (on half the branches, the others ending in a leaf at depth n - 1) and have cost:

$$E(T) = n - 1 + 2^{n-2}\cdot(\gamma + 1)\cdot\varepsilon ,$$

while the optimal trees test x_1 first and have cost:

$$E(T_o) = 1 + (n - 1)\cdot 2^{n-1}\cdot\varepsilon .$$

(The case n = 4 is illustrated in Figure 5.4.) Thus, if $\varepsilon << 1$, (e.g., if $\varepsilon = 2^{-kn}$ for some $k > 1$), the asymptotic ratio of costs becomes

$$C(T)/C(T_o) \simeq n - 1 .$$

By letting every point satisfying

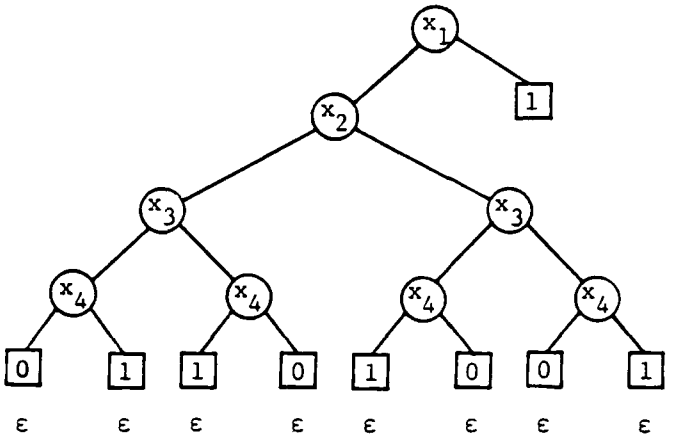
$$x_1 \cdot \bigoplus_{i=2}^{n} x_i = 1$$

be mapped to a recursive call, the worst case for completely specified recursive Boolean functions is obtained. In that case, the best diagram, D_o, has a cost of $(1 + (n - 1)\cdot 2^{n-1}\cdot\varepsilon)/(1 - 2^{n-2}\cdot\varepsilon)$ while the diagrams constructed by the loss criterion, D, have a cost of $n/(1 - 2^{n-2}\cdot\varepsilon)$; thus, the asymptotic ratio, $E(D)/E(D_o)$, becomes approximately n for small ε. (That both recursive and nonrecursive cases yield the same worst case, $\mathcal{O}(n)$, is due to the fact that the recursive factor, 1/e, is independent of tree structure and gets factored out in the ratio.)

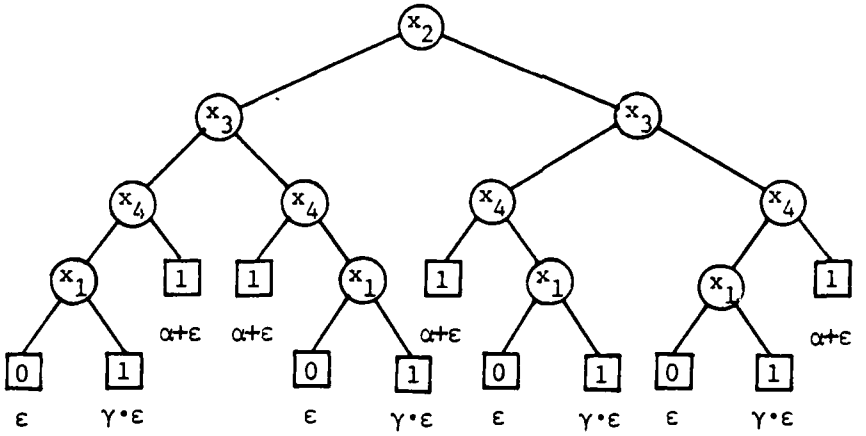
In this example, however, the lower bound set by Corollary 5.1 remains arbitrarily close to the optimal cost, since:

$$lb_f(x_i) = 1 + (n - 2)\cdot 2^{n-1}\cdot\varepsilon + 2^{n-2}\cdot(\gamma + 1)\cdot\varepsilon, \; i \neq 1 ,$$

so that $E(T_o) - lb_f(x_i) = 2^{n-2} \cdot (1 - \gamma) \cdot \varepsilon \simeq 0$. Thus, the pessimal behavior of the loss criterion could be detected at an early stage in the construction of the tree and the selections revised. This is not to say that the lower bound set by Corollary 5.1 always remains close to



(a)



(b)

Figure 5.4. The two trees, with their leaf probabilities, for the worst case performance of the loss criterion when n = 4: (a) the optimal tree, T_o, and (b) the pessimal tree, T.

the optimal expected testing cost: it is easy to construct an identification problem with n binary variables of unity cost and 2^{n-1} equally likely objects so that the lower bound is always 1 while the optimal cost is n - 1 (it is enough to have each object characterized by a combination of test values, the sum of which is equal to 1 modulo 2).

It must be noted, however, that there is little use for any selection heuristics, since an equally efficient optimal algorithm is available [the dynamic programming algorithm of (Bayes 73, Schumacher 76)], which can easily be extended to apply to recursive relations. Nevertheless, a selection criterion may be useful in that it indicates the importance of individual variables; in particular, the activity has potential applications as a measure of complexity (further investigated in the next section), as a tool for system testing (which application is the subject of Chapter 6), and as a gauge of the power of discrimination of a variable in data base queries, pattern recognition, and other decision problems.

5.5. The Intrinsic Cost as a Measure of Complexity

As indicated in Chapter 1, current measures of the complexity of discrete functions are not entirely satisfying, mostly because of their dependence on some form of implementation. Thus, software science (Halstead 77) studies the complexity of computer programs as perceived by humans, combinational complexity (Savage 76, Pippenger 77) measures the complexity of Boolean functions in terms of the number of gates in a circuit or the number of terms in a Boolean formula, and even concrete

computational complexity (Aho 74, Garey 79) uses a computer-like model on which space and time measures are developed. Other measures [a recent and succinct survey of which can be found in (Rouse 79)] are even more specialized.

The pitfalls of an approach keyed to a particular mode of implementation are vividly illustrated by a comparison between the results of combinational complexity and those of (Lee 59), both about Boolean functions. Combinational complexity is concerned with the number of gates (each realizing one of the sixteen Boolean functions of two variables) necessary to compute a function; for a function of n variables, it is known that $\Theta(2^n/n)$ gates are needed when subfunctions can be regrouped, and $\Theta(2^n/\log n)$ when they cannot (as in representation by unfactored Boolean formulae). Lee's results show that a function of n variables can be represented with $\Theta(2^n/n)$ nodes by a decision diagram; this is equal to the necessary number of gates, but only $\Theta(n)$ operations are needed in decision diagrams versus $\Theta(2^n/\log n)$ in a combinational network. This clearly indicates that combinational complexity is inadequate as an implementation-independent measure of complexity and that, moreover, decision trees and diagrams capture some essential property of discrete functions that is not reflected in some other measures.

Concrete complexity theory, as mentioned in Chapter 1, has made use of decision trees in order to derive lower bounds on the computational complexity of classes of problems. However, the approach taken (measuring the worst case testing cost) is directly dependent on the

decision tree model and, as seen in Section 4.3.3, the results are mostly trivial since almost all Boolean functions are exhaustive.

In conclusion, then, a measure is desired which captures some of the aspects of decision trees but does not directly depend on them. Such is the intrinsic cost and its use as a complexity measure for discrete functions is proposed here. The intrinsic cost is implementation-independent; it is a lower bound on the cost of any type of evaluation of a function. Being the sum of activities, it also lends itself to an analysis of the influence of individual variables; similarly, since the activity is compatible with the process of decomposition, the intrinsic cost of subfunctions is readily computed from that of the given function. Both properties are, of course, indispensable for a system design and analysis tool.

Validating an applied complexity measure is a large undertaking, beyond the present scope of this investigation. It is felt, nevertheless, that the results available in the literature and presented in this work justify the need for a complexity measure such as the intrinsic cost. Further justification may be found in the close relationship between activity and some problems of testing, which relationship is discussed in the next chapter.

CHAPTER 6

APPLICATION TO SYSTEM TESTING

6.1. Introduction

Decision trees have been used by numerous researchers for purposes of machine diagnosis (Hoehn 58, Brulé 60, LaMacchia 62, Chang 70, Koren 77); in most cases, however, the tree is the specification of a testing algorithm. More recently (Akers 78a, 78b) proposed to use Boolean graphs (that he called decision diagrams) as representations of Boolean functions in order to develop testing schemes. He also included sequential systems by considering only their next state function and using decomposition to the point where, it was reasoned, any given part of the analysis encounters few next state variables.

A different method is adopted here. In keeping with modern system architecture, in particular the use of large scale integration, it is assumed that a system has many more internal variables than can reasonably be controlled (or even examined) in a testing experiment. The emphasis is placed on signal reliability (Koren 79), that is, a measure of the probability that the system's output is correct, rather than the conventional functional reliability, a more pessimistic measure based on the probability that a system is fault free. Combinational systems are first examined and a relation is established between the activity of a variable and the probability of detecting a faulty output. The results are then extended to sequential systems using a steady state model. Finally, further applications are briefly discussed.

6.2. Combinational Systems

A combinational system has no internal memory and is entirely described by its output relation (according to the premises stated in Chapter 1). Figure 6.1 illustrates such a system, F, with n input variables and an output specified by a relation, R, on the input variables.

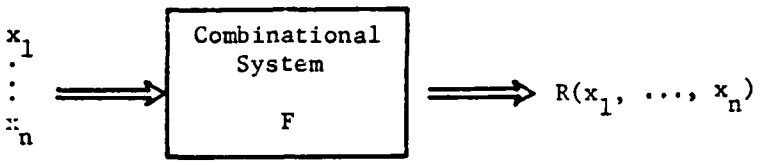


Figure 6.1. A combinational system.

To test a newly produced or previously used version of F by verifying its outputs while all input vectors are applied--a method called exhaustive testing--is a task, the complexity of which grows exponentially with the number of variables; accordingly, exhaustive testing quickly becomes impractical. This has prompted the study of alternate approaches, which do not guarantee the detection of every fault (only exhaustive testing can), but minimize the probability that an error will go undetected. In particular (Losq 76, 78) has shown that random compact testing is effective for large scale systems realizing combinational or sequential functions. In this method, all inputs are controlled; a sequence of random input vectors is applied and output statistics (such as the frequency of occurrence of each value) are

gathered. The test statistics are then compared with the output statistics of a perfect unit (the "gold unit") and the unit under test is accepted as being in working order if its output statistics fall within an "acceptance window" around those of the gold unit. (Losq 78) showed that, when all faults are assumed to be permanent (i.e., lines "stuck-at" a particular value), the probability of acceptance of a faulty unit approaches zero as the test length increases and is very small already for tests that are much shorter than exhaustive sequences.

In large systems in operation, it is not always desirable (or even feasible) for a test generator to assume control of all input variables. Several variables may be altogether unavailable (for instance, the secondary variables of a sequential circuit) while the value of others cannot be modified during operation. In such a case, one is reduced to controlling a subset of the inputs and allow the others to vary. A test consists then of an exhaustive application of all the combinations of controlled inputs and a verification of the output for each input vector.

Permanent faults are assumed; since the system's implementation is unknown, only input-output relationships can be observed and thus any test will measure signal reliability. Specifically, a fault is assumed to cause the system output to be stuck at a given value at a time when a gold unit would produce a different result. Let the system, F, have the output set, Ω, and, in first approximation of the fault model, let A_i, $i = 1, \ldots, |\Omega|$, be the event that a failure causes the system's output to be stuck at the i-th output value. Since those events are

disjoint, the probability, p_e, that the system's output is incorrect is

$$p_e = p\left(\bigcup_{i=1}^{|\Omega|} A_i\right) = \sum_{i=1}^{|\Omega|} p(A_i) .$$

If all events are equally likely, with probability $p(A_i) = \delta$, then

$$p_e = |\Omega| \cdot \delta$$

and the probability, p_c, that the output is correct is

$$p_c = 1 - |\Omega| \cdot \delta .$$

Now, let the set of controlled variables be composed of the single variable, x_i; the other variables are allowed to vary according to the known probability distribution over input variable values. [Essentially, the remaining inputs are assumed to arise from a zero-th order Markov process (Parzen 62), in which each set of values is selected according to fixed probabilities independently of preceding values.] The test sequence accordingly consists of all m_i values that x_i can assume. An incorrect output will be observed during the test exactly if either

(a) a fault is present and the relation is sensitized to x_i, or

(b) the relation is not sensitized to x_i but is stuck at a value other than any that could correctly result from the remaining (n - 1) variables' values.

These are disjoint events, the first occurring with probability

$$p_a = p_R^+(x_i) \cdot p_e$$

and the second with probability

$$p_b = \sum_{C_j} p(C_j) \cdot p\left(\bigcup_k A_k\right) ,$$

where the union is taken over all events, A_k, such that $R(x_k) \cap R(x_j) = \phi$ and the sum is taken over all combinations, C_j, of the remaining n - 1 variables such that R *is not sensitized to* x_i. *If R is a completely* specified function and the failure events are equally likely, those probabilities become:

$$p_a = p_R^+(x_i) \cdot |\Omega| \cdot \delta ,$$

and

$$p_b = p_R^-(x_i) \cdot (|\Omega| - 1) \cdot \delta ,$$

so that the probability of observing an incorrect output by applying all possible values of variable x_i is

$$\begin{aligned} p_a + p_b &= p_R^+(x_i) \cdot |\Omega| \cdot \delta + p_R^-(x_i) \cdot (|\Omega| - 1) \cdot \delta \\ &= |\Omega| \cdot \delta \cdot \left(p_R^+(x_i) + p_R^-(x_i)\right) - \delta \cdot p_R^-(x_i) \\ &= |\Omega| \cdot \delta - \delta \cdot p_R^-(x_i) . \end{aligned}$$

Since $p_R^+(x_i) = a_R(x_i)$ when costs are unity, it follows that testing that variable which has the highest activity (i.e., the lowest loss) maximizes the probability of detecting an incorrect output value.

Theorem 6.1. Under the above assumptions, the probability of detecting an incorrect output value, p_d, obeys the following inequalities:

p_d(using the highest activity variable) $\geq$

p_d(using randomly selected variable) $\geq$

p_d(using lowest activity variable) . □

The above results extend in a natural way to controlling subsets of k out of the n variables.

6.3. Extension to Sequential Systems

A sequential system incorporates memory. The standard model of a discrete parameter system with memory distinguishes the memory unit and the combinational subsystem, as illustrated in Figure 6.2. The system's output is a relation on the primary (external) variables, $x_1, \ldots, x_n$, and the secondary (internal) variables, $y_1, \ldots, y_m$. [When the system is Boolean, this model is known as a Mealy machine (Friedman 75).] It is often the case in practice that the memory unit values, $y_1, \ldots, y_m$, are not directly controllable or even observable. For instance, the limited number of pins available on packages for large scale integrated circuits does not usually allow the allocation of any connections to the secondary input variables. It is therefore assumed that only the primary

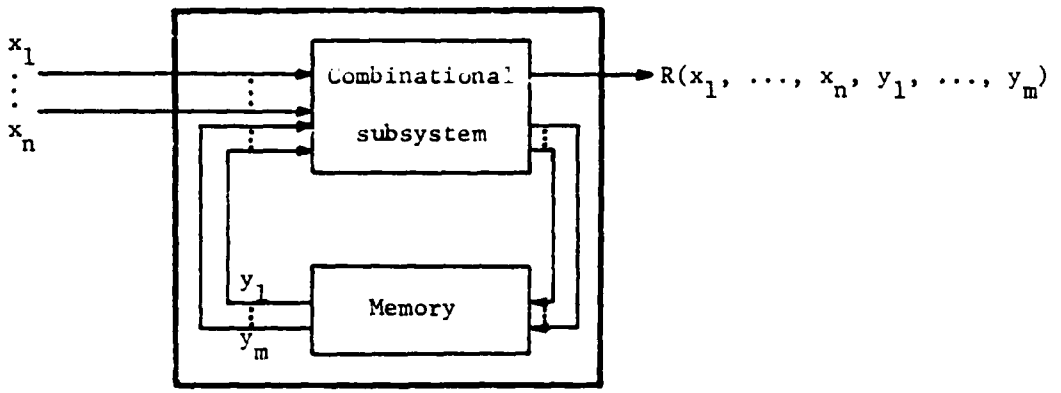


Figure 6.2. The conventional model of a sequential system.

variables can be controlled, even for purposes of testing; further, it is assumed that a fixed probability distribution over the vectors of values of the primary inputs is known, which represents the normal operating conditions of the system.

Since the system is discrete and finite, it can assume only a finite number of states, one for each combination of values of the secondary variables. The probability, p_{ij}, of transition from state i to state j can then be computed from the system's known relations and the probability distribution of the primary variables. Thus, the system can be considered as a Markov chain (Parzen 62, Booth 67), which is irreducible (since every state can be reached from every other state) and recurrent positive (since the system eventually returns to each state with a probability of one). This is a stochastic process about which, in particular, the following results are known.

(a) There is convergence in Cesàro mean to a vector of values that can be interpreted as the probability distribution over the states in the long run; that is, the i-th element, π_i, in that vector is the fraction of time that, in the limit, the system spends in state i. The elements of the vector can be computed by a set of linear equations of the form

$$\pi_i = \sum_j \pi_j \cdot p_{ji} .$$

(b) Let $N_i(n)$ be the number of times that the Markov chain is in state i during its visits to the first n states; then this occupation time of i is asymptotically gaussian with expectation $n \cdot \pi_i$ and variance $n \cdot \sigma_{ii}^2 \cdot \pi_i^3$, where σ_{ii}^2 is the variance of the random variable describing the recurrence time of state i.

The first property implies that, in the long run, a fixed probability distribution on the primary input values induces a fixed distribution on the secondary variables' values (the states). This allows the computation of the activities of the primary variables; long run testing by controlling only the primary variables (and not letting the test sequences alter the predetermined probability distribution) can then be carried out in a way similar to the testing of combinational systems described in the previous section, with the same results.

6.4. Discussion

The results presented in the previous sections are, of course, only preliminary. In particular, the relationship between activity and the

probability of error detection must be worked out in the more general case of nonuniform failure probabilities, and the fault model itself must be validated.

As a first step toward the latter goal, it must be noted that the fault model adopted in this chapter, which assumes that only inputs and outputs (i.e., external signals) are observable, is better suited to modern methods of system implementation, especially large scale integration, than the conventional model of functional reliability. Given a system implemented by a single very large scale integrated (VLSI) circuit, the engineer is concerned about its proper functioning, that is, a correct input-output behavior; if an error is detected, the whole package is replaced: the nature of the problem inside the package is of no concern.

When testing a combinational circuit "on the bench" (that is, not in operation), all of the inputs are usually controllable, so that testing only part of the variables may be an unnecessary restriction; even if exhaustive testing is not permissible, random compact testing can be used. While the latter is certainly indicated, there may yet be reasons for which an exhaustive test of the variables with highest activities would be preferred. In particular, such a test thoroughly checks out the behavior of the system with respect to the most "significant" variables; as such, a successful test indicates that the functionally important parts of the system are in working order. A measure of the extent of these parts is then the ratio of the sum of the

activities of the exhaustively tested variables to the intrinsic cost of the system.

This property has potential applications to the quality control of VLSI circuits, where the yield of perfect circuits is often low. One could conceive a test conducted on the most relevant variables in order to guarantee that at least the important parts of the circuit are in working order; those imperfect circuits that passed the test could then be released for noncritical applications, rather than altogether discarded. (This could lower the price of VLSI production; for further recycling of costly products, the process could be coupled with limited repair capabilities in order to bring more circuits within the predetermined percentage of the intrinsic cost necessary for acceptance.)

Yet other applications are foreseeable. Clearly, however, a good deal of prior research is necessary.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

The use of decision trees and diagrams as models of discrete functions has been investigated. A general framework has been elaborated, into which the diverse results available in the literature have been brought. In particular, a complete analysis of the complexity of optimization of decision trees has been presented, including several new results on the worst case computational complexity of Boolean functions. After a discussion of the various measures defined on decision trees and diagrams, a single measure, the expected testing cost, was selected as representative of the complexity of decision trees. This measure was further examined in order to develop a measure of complexity on functions. In particular, the concept of the activity of a variable, a measure of the contribution of a variable to the testing cost of a function, was introduced, and its relation to decision trees was established. These concepts were shown to be applicable to recursive relations and hierarchies of relations.

After a brief discussion of the applications of activity to the construction of decision trees, its use as a complexity measure for discrete functions was proposed and discussed. Finally, the application of activity to problems of testing was examined, with particular emphasis on the testing of large scale integrated systems in operation. Further research is needed in order to validate the proposed measure of

complexity and develop some specific characterizations. Although the testing applications introduced in the previous chapter are of a preliminary nature, they clearly indicate the potential of the concept of activity for solving some of the acute problems encountered in testing large systems.

REFERENCES

REFERENCES

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass.

Akers, S. B. (1978a). Binary decision diagrams. IEEE Trans. Comp. TC-27, 6, 509-516.

Akers, S. B. (1978b). Functional testing with binary decision diagrams. Proc. 8th Fault-Tolerant Comp. Symp., Toulouse.

Bayes, A. J. (1973). A dynamic programming algorithm to optimize decision table code. Austral. Comp. J. 5, 2, 77-79.

Blum, M., Chandra, A. K., and Wegman, M. N. (1980). Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. Inf. Proc. Letters 10, 2, 80-82.

Booth, T. L. (1967). Sequential Machine and Automata Theory. John Wiley & Sons, New York.

Bozoyan, Sh. Ye. (1975). Certain properties of Boolean differentials and the activities of the arguments of Boolean functions and questions of construction of reliable schemes that have unreliable elements. Eng. Cybernetics 5, 108-120 (transl. from Russian).

Breitbart, Y., and Gal, S. (1978). Analysis of algorithms for the evaluation of monotonic Boolean functions. IEEE Trans. Comp. TC-27, 11, 1083-1087.

Breitbart, Y., and Reiter, A. (1975a). Algorithms for fast evaluation of Boolean expressions. Acta Inf. 4, 107-116.

Breitbart, Y., and Reiter, A. (1975b). A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic Boolean functions. Acta Inf. 4, 311-319.

Brulé, J. D., Johnson, R. A., and Kletsky, E. J. (1960). Diagnosis of equipment failures. IRE Trans. Rel. and Ctrl RQC-9, 23-34.

Cerny, E., Mange, D., and Sanchez, E. (1979). Synthesis of minimal binary decision trees. IEEE Trans. Comp. TC-28, 7, 472-482.

Chang, H. Y. (1965). An algorithm for selecting an optimum set of diagnostic tests. IEEE Trans. Comp. EC-14, 5, 706-711.

Chang, H. Y., Manning, E., and Metze, G. (1970). Fault Diagnosis of Digital Systems. John Wiley & Sons, New York, Chapter 3.

Davio, M., and Thayse, A. (1977). Sequential evaluation of Boolean functions. MBLE Res. Lab. Rep. R341, Brussels.

Fortune, S., Hopcroft, J. E., and Schmidt, E. M. (1978). The complexity of equivalence and containment for free single variable program schemes. Lecture Notes in Comp. Sc. (Springer, Berlin), 227-240.

Friedman, A. D., and Menon, P. R. (1975). Theory and Design of Switching Circuits. Computer Science Press, Potomac, Md.

Ganapathy, S., and Rajamaran, V. (1973). Information theory applied to the conversion of decision tables to computer programs. Comm. ACM 16, 9, 532-539.

Garey, M. R. (1972). Optimal binary identification procedures. SIAM J. Appl. Math. 23, 2, 173-186.

Garey, M. R. (1980). Private communication.

Garey, M. R., and Graham, R. L. (1974). Performance bounds on the splitting algorithm for binary testing. Acta Inf. 3, 347-355.

Garey, M. R., and Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, San Francisco.

Gower, J. C., and Barnett, J. A. (1971). Selecting tests in diagnostic keys with unknown responses. Nature 232, 491-493.

Green, C. (1973). A path entropy function for rooted trees. J. ACM 20, 3, 378-384.

Gyllenberg, H. G. (1963). A general method for deriving determination schemes for random collections of microbial isolates. Ann. Acad. Sc. Fenn. A IV, 69, 1-23.

Halpern, J. (1974a). Evaluating Boolean functions with random variables. Int. J. Syst. Sc. 5, 6, 454-553.

Halpern, J. (1974b). A sequential testing procedure for a system's state identification. IEEE Trans. Rel. R-23, 4, 267-272.

Halstead, M. H. (1977). Elements of Software Science. Elsevier North Holland, New York.

Hanani, M. Z. (1977). An optimal evaluation of Boolean expressions in an online query system. Comm. ACM 20, 5, 344-347.

Harary, F. (1969). Graph Theory. Addison-Wesley, Reading, Mass.

Harrison, M. A. (1965). Introduction to Switching and Automata Theory. McGraw-Hill, New York.

Hoehn, A. J., and Saltz, E. (1958). Mathematical models for determination of efficient troubleshooting routes. IRE Trans. Rel. and Ctrl PGRQC-13, 1-14.

Huffman, D. A. (1952). A method for the construction of minimum redundancy codes. Proc. IRE 40, 1098-1101.

Hung, T. Q. (1974). La construction des clés de diagnostic. M.S. Thesis, University of Montreal.

Hyafil, L., and Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. Inf. Proc. Letters 5, 1, 15-17.

Ianov, I. (1960). The logical schemes of algorithms. In Problems and Cybernetics, Vol. 1, Pergamon Press, New York, 82-140.

Jardine, N., and Sibson, R. (1971). Mathematical Taxonomy. John Wiley & Sons, New York.

Johnson, S. M. (1956). Optimal sequential testing. Project RAND Res. Mem. RM-1652.

Karp, R. M. (1972). Reducibility among combinational problems. In R. E. Miller and J. W. Thatcher (eds.), Complexity of Computer Computations, Plenum Press, New York, 85-103.

King, P.J.H., and Johnson, R. G. (1973). Some comments on the use of ambiguous decision tables and their conversion to computer programs. Comm. ACM 16, 5, 287-290.

King, P.J.H., and Johnson, R. G. (1974). Comments on the algorithms of Verhelst for the conversion of limited-entry decision tables to flowcharts. Comm. ACM 17, 1, 43-44.

Kletsky, E. J. (1960). An application of the information theory approach to failure diagnosis. IRE Trans. Rel. and Qual. Ctrl RQC-9, 29-39.

Knuth, D. E. (1971). Mathematical analysis of algorithms. Proc. IFIP Congress 71, Vol. I, 135-143.

Knuth, D. E. (1973). The Art of Computer Programming. Volume 1: Fundamental Algorithms. Addison-Wesley, Reading, Mass. (2nd ed.).

Koren, I., and Kohavi, Z. (1977). Sequential fault diagnosis in combinational networks. IEEE Trans. Comp. TC-26, 4, 334-342.

Koren, I. (1979). Analysis of the signal reliability measure and an evaluation procedure. IEEE Trans. Comp. TC-28, 3, 244-249.

LaMacchia, S. E. (1962). Diagnosis in automatic checkout. IRE Trans. Mil. Elec. MIL-6, 302-309.

Lee, C. Y. (1959). Representation of switching circuits by binary-decision programs. Bell Syst. Tech. J. 38, 985-999.

Losq, J. (1976). Referenceless random testing. Proc. 6th Int. Symp. Fault-Tolerant Comp., Pittsburgh, 108-113.

Losq, J. (1978). Efficiency of random compact testing. IEEE Trans. Comp. TC-27, 6, 516-525.

Loveland, D. W. (1979). Selecting optimal test procedures from incomplete test sets. Duke Univ. Tech Rep. CS 1979-7.

Mandelbaum, D. (1964). A measure of efficiency of diagnostic tests upon sequential logic. IEEE Trans. Comp. EC-13,630.

Mange, D., and Sanchez, E. (1978). Synthèse des fonctions logiques avec des multiplexeurs. Digital Proc. 4, 29-44.

Martelli, A., and Montanari, U. G. (1978). Optimizing decision trees through heuristically guided search. Comm. ACM 21, 12, 1025-1039.

Masek, W. J. (1980). Some NP-complete set covering problems. To appear in Theoretical Comp. Sc.

Meisel, S. W., and Michalopoulos, D. A. (1973). A partitioning algorithm with application in pattern classification and the optimization of decision trees. IEEE Trans. Comp. TC-22, 1, 93-103.

Metzner, J. R., and Barnes, B. H. (1977). Decision Table Languages and Systems. Academic Press, New York.

Michalski, R. S. (1978). Designing extended-entry decision tables and optimal decision trees using decision diagrams. Univ. of Illinois at Urbana-Champaign Tech. Rep. UIUCDCS-R-78-898 (also IEEE Comp. Rep. R78-126).

Miller, R. E., Pippenger, N., Rosenberg, A. L., and Snyder, L. (1979). Optimal 2, 3-trees. SIAM J. Comp. 8, 1, 42-59.

Misra, J. (1972). A study of strategies for multistage testing. Ph.D. Thesis, The Johns Hopkins University.

Montalbano, M. (1962). Tables, flow charts, and program logic. IBM Syst. J. 1, 51-63.

Morse, L. E. (1971). Specimen identification and key construction with time-sharing computers. Taxon 20, 269-282.

Osborne, D. V. (1963). Some aspects of the theory of dichotomous keys. New Phytol. 62, 111-160.

Pankhurst, R. J. (1970). A computer program for generating diagnostic keys. Comp. J. 13, 2, 145-151.

Parzen, E. (1962). Stochastic Processes. Holden-Day, San Francisco.

Payne, H. J., and Meisel, W. S. (1977). An algorithm for constructing optimal binary decision trees. IEEE Trans. Comp. TC-26, 9, 905-916.

Perl, Y., and Breitbart, Y. (1976). Optimal sequential arrangement of evaluation trees for Boolean functions. Inf. Sc. 11, 1-12.

Picard, C. F. (1972). Graphes et Questionnaires. Volume 2: Questionnaires. Gauthier-Villars, Paris.

Pippenger, N. (1977). Information theory and the complexity of Boolean functions. Math. Syst. Theory 10, 129-167.

Pollack, S. L. (1965). Conversion of limited-entry decision tables to computer programs. Comm. ACM 8, 11, 677-682.

Pooch, U. W. (1974). Translation of decision tables. Comp. Surveys 6, 125-151.

Prather, R. E., and Casstevens, H. T. (1978). Realization of Boolean expressions by atomic digraphs. IEEE Trans. Comp. TC-27, 8, 681-688.

Press, L. I. (1965). Conversion of decision tables to computer programs. Comm. ACM 8, 6, 385-390.

Rabin, J. (1971). Conversion of limited-entry decision tables into optimal decision trees: fundamental concepts. SIGPLAN Notices 6, 8, 68-74.

Reingold, E. M. (1972). On the optimality of some set algorithms. J. ACM 19, 4, 649-659.

Reinwald, L. T., and Soland, R. M. (1966). Conversion of limited-entry decision tables to optimal computer programs I: minimum average processing time. J. ACM 13, 3, 339-358.

Reinwald, L. T., and Soland, R. M. (1967). Conversion of limited-entry decision tables to optimal computer programs II: minimum storage requirements. J. ACM 14, 4, 742-755.

Rescigno, A., and Maccacaro, G. A. (1961). The information content of biological classifications. In Information Theory: Fourth London Symposium, C. Cherry, ed., Butterworths, London, 437-445.

Riesel, H. (1963). In which order are different conditions to be examined. BIT 3, 255-256.

Rivest, R. L., and Vuillemin, J. (1976a). On the number of argument evaluations required to compute Boolean functions. U.C. Berkeley Elec. Res. Lab. Mem. ERL-M476.

Rivest, R. L., and Vuillemin, J. (1976b). On recognizing graph properties from adjacency matrices. Theoretical Comp. Sc. 3, 371-384.

Rounds, E. M. (1979). A combined nonparametric approach to feature selection and binary tree design. Proc. Conf. Pattern Rec. and Image Proc., Chicago, 38-43.

Rouse, W. B., and Rouse, S. H. (1979). Measures of complexity of fault diagnosis tasks. IEEE Trans. Syst., Man, and Cybern. SMC-9, 11, 720-727.

Savage, J. E. (1976). The Complexity of Computing. John Wiley & Sons, New York.

Schumacher, H., and Sevcik, K. C. (1976). The synthetic approach to decision table conversion. Comm. ACM 19, 6, 343-351.

Sethi, I. K., and Chatterjee, B. (1977). Efficient decision tree design for discrete variable pattern recognition problems. Pattern Rec. 9, 197-206.

Sethi, I. K., and Chatterjee, B. (1980). Conversion of decision tables to efficient sequential testing procedures. Comm. ACM 23, 5, 279-285.

Shwayder, K. (1971). Conversion of limited-entry decision tables to computer programs--a proposed modification of Pollack's algorithm. Comm. ACM 14, 2, 69-73.

Shwayder, K. (1974). Extending the information theory approach to converting limited-entry decision tables to computer programs. Comm. ACM 17, 9, 532-537.

Shwayder, K. (1975). Combining decision rules in a decision table. Comm. ACM 18, 8, 476-480.

Slagle, J. R. (1964). An efficient algorithm for finding certain minimum cost procedures for making binary decisions. J. ACM 11, 3, 253-264.

Slagle, J. R., and Lee, R.C.T. (1971). Application of game tree searching techniques to sequential pattern recognition. Comm. ACM 14, 2, 103-110.

Standish, T. A. (1980). Data Structure Techniques. Addison-Wesley, Reading, Mass. (Section 4.4).

Thayse, A., and Davio, M. (1973). Boolean differential calculus and its application to switching theory. IEEE Trans. Comp. TC-22, 4, 409-420.

Thayse, A., Davio, M., and Deschamps, J. P. (1978). Optimization of multivalued decision algorithms. Proc. Int. Symp. on Multival. Logic, Rosemont, 171-178.

Ullman, J. D. (1980). Principles of Database Systems. Comp. Sc. Press, Potomac, Md.

Verhelst, M. R. (1972). The conversion of limited-entry decision tables to optimal and near-optimal flowcharts: two new algorithms. Comm. ACM 15, 11, 974-980.

Willcox, W. R., and Lapage, S. P. (1972). Automatic construction of diagnostic tables. Comp. J. 15, 263-267.

Winder, R. O. (1971). Chow parameters in threshold logic. J ACM 18, 2, 265-289.

Winston, P. (1969). A heuristic program that constructs decision trees. Artif. Intell. Mem. 173, Project MAC, M.I.T.

Yasui, T. (1971). Some aspects of decision table conversion techniques. SIGPLAN Notices 6, 8, 104-111.

Yasui, T. (1972). Conversion of decision tables into decision trees. Ph.D. Thesis, UIUCDCS-R-72-501, Univ. of Illinois as Urbana-Champaign.

You, K. C., and Fu, K. S. (1976). An approach to the design of a linear binary tree classifier. Proc. Symp. on Machine Proc. of Remotely Sensed Data, Purdue, 3A1-3A10.

Zimmerman, S. (1959). An optimal search procedure. Amer. Math. Monthly 66, 690-693.


TE
MED